RasPi

DESIGN
BUILD
CODE

29

Get hands-on with your Raspberry Pi

STREAM INTERNET TV TO YOUR PI

+ BUILD AN iBEACON

Plus Capture night photos with Pi NoIR

Welcome

How much TV do you watch through an old-fashioned provider and how much do you watch online? If you're anything like us, your viewing habits are leaning more and more towards box set binges than teatime telly. Want something that makes it easier to stream your favourite online content direct to your TV? We'll show you how to use the Pi to do exactly that.

Also this issue you'll learn how to build an iBeacon, take night photos with the Pi NoIR camera, continue making your own version of retro game Tempest in FUZE Basic, and add multitasking functionality to your Python code. Whatever you want to do with your Pi, we've got you covered. Enjoy the issue!

Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

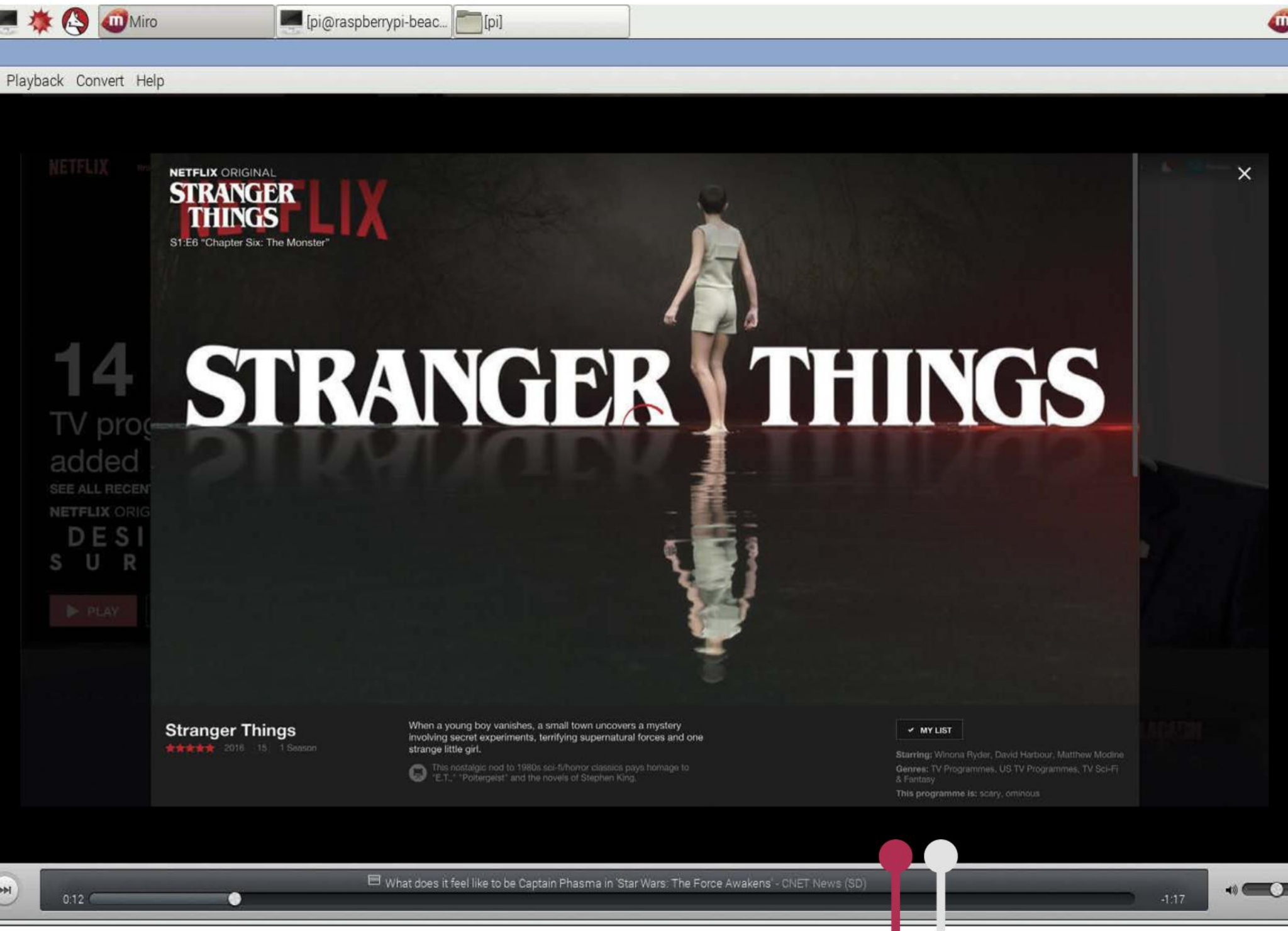
Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk

Contents

Stream internet TV to your Raspberry Pi

Get your favourite shows and video podcasts streamed automatically to your TV with Miro



Finding the content you're interested in viewing can take a while. Whether you're looking for internet TV stations, video podcasts, audio podcasts or shows syndicated online, taking the time to find and download them can be slow going, particularly if you have a busy lifestyle. You might even have no time to watch after you've waited for the download.

Thanks to the Miro media management software, we can automate all of this, and with the software running on a Raspberry Pi, you can easily build a compact system for downloading and playing back shows that you have an interest in. We're talking targeted TV on demand, which makes this project ideal for staying up to date with particular news and trends on a certain topic.

THE PROJECT ESSENTIALS

Raspbian Wheezy
HDMI cable
Monitor/TV display

01 Set up your Pi with Raspbian

Sadly, Miro cannot run on Raspbian Jessie, so make sure you're using Wheezy, available via raspberrypi.org/downloads/raspbian. Ensure your Pi is connected to a TV or display via HDMI.

As Miro is a desktop application, you'll need your mouse and keyboard connected to configure it.

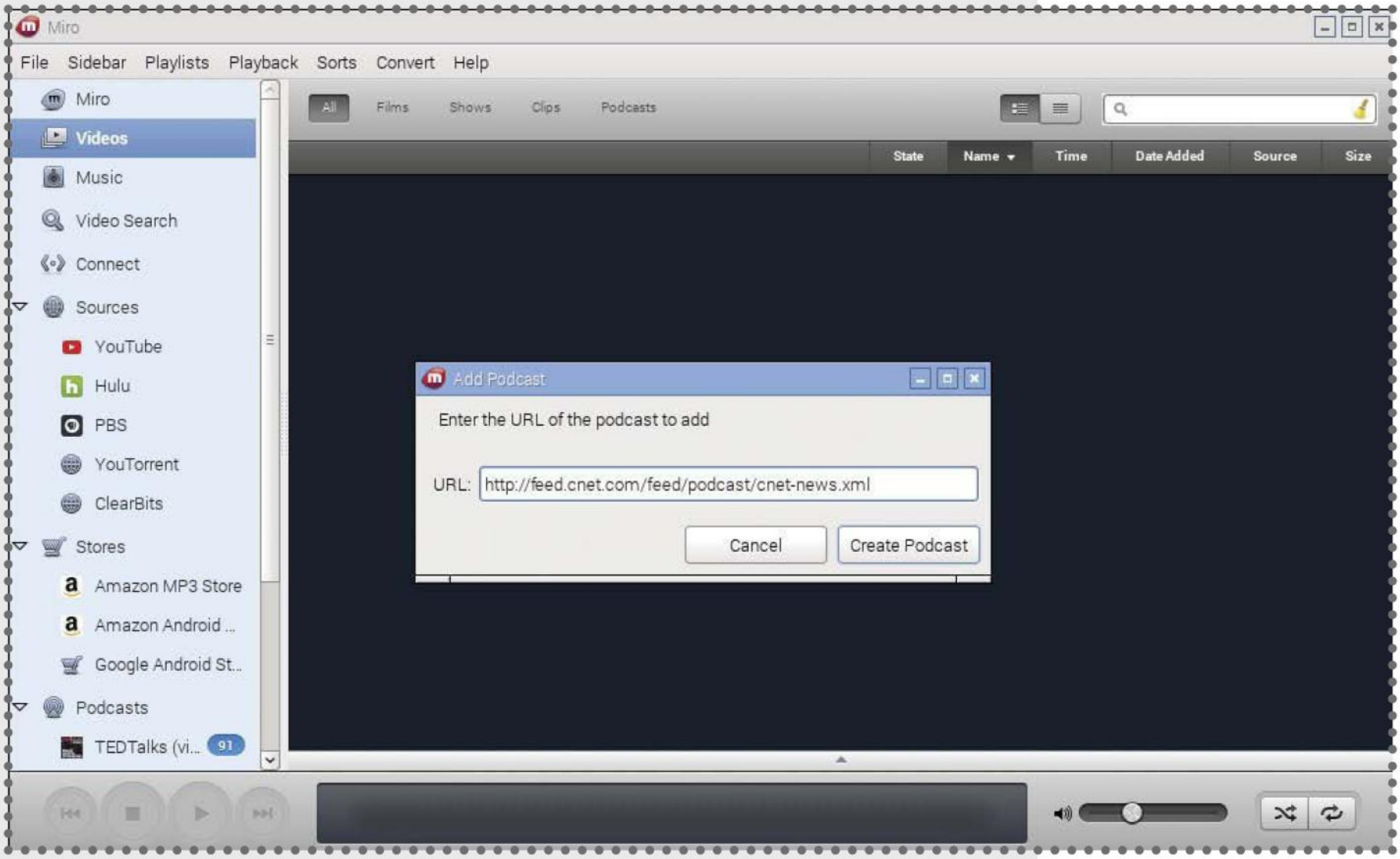
02 Install Miro

With Wheezy flashed to your SD card and your Pi booted up, open Terminal and enter:

```
sudo apt-get install miro
```

Installation will take a few moments. Once complete, you'll find Miro in Menu>Sound and Video. Click it to get started.

"The more links you add, the more regularly updated content will be downloaded to your media manager"



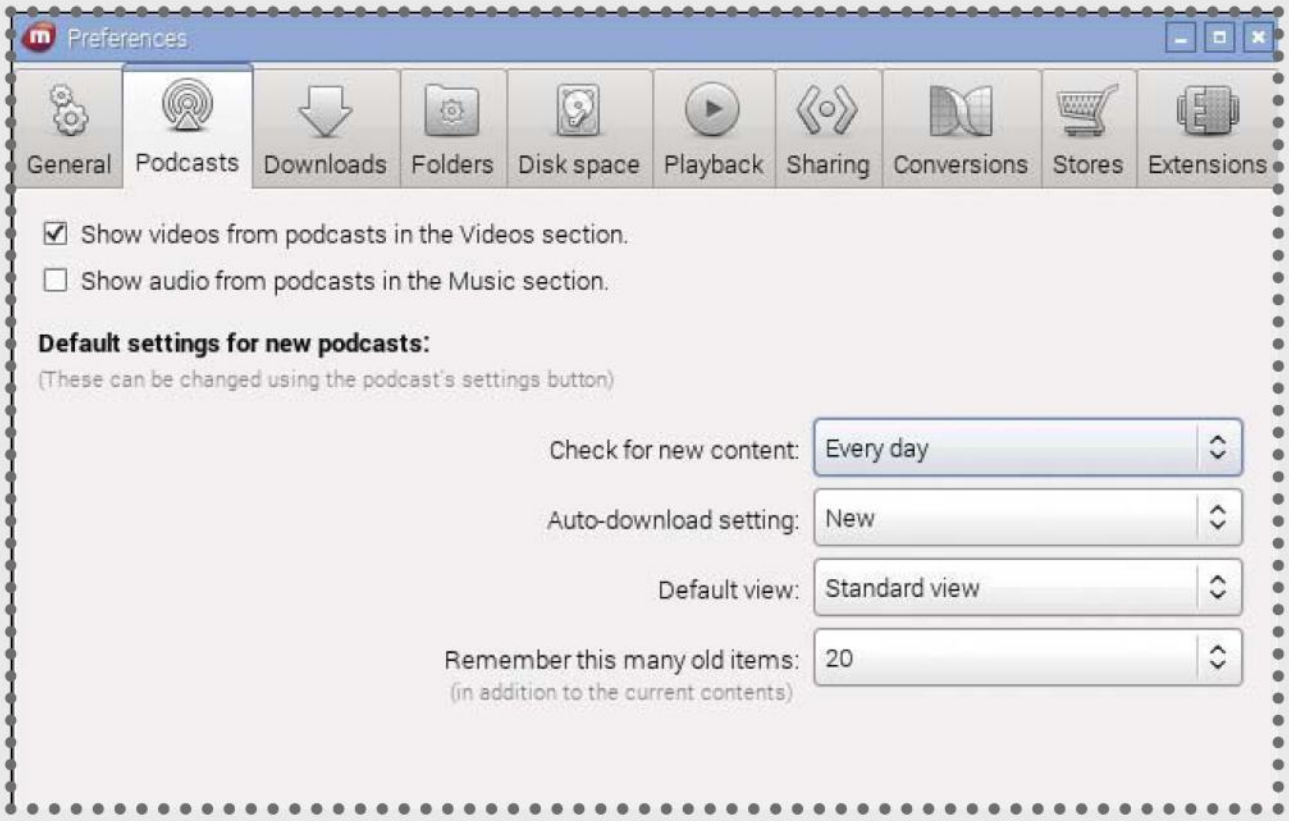
03 Set Miro to launch at startup

Make sure Miro app is configured to launch at startup. Open File>Preferences>General and check 'Automatically run Miro when I log in' and 'When starting up Miro remember what screen I was on when I last quit'. Also set your Pi to boot into X using the raspi-config utility.

04 Check for content

Switch to the Podcasts tab and place a check in the box labelled 'Show videos from podcasts in the Videos section'. On the right-hand side of the window, set your preferred frequency for checking for new content. Miro will poll your favourite websites and feeds based on this setting.

Above You can subscribe to all sorts of content, from internet TV channels to news podcasts



05 Configure playback settings

Move now to the Playback tab, and check Play media in Miro. This limits reliance on other apps, which may drain resources. You should also click the Play video and audio items one after another radio button, and under Resume Playback, check the first and third items.

06 Source videos and podcasts

With Miro set up to play back the video and audio content you want to enjoy, it's time to find some! The best way to do this is to just check the websites that you regularly use for video and audio podcasts (preferably the former) and then copy the XML link.

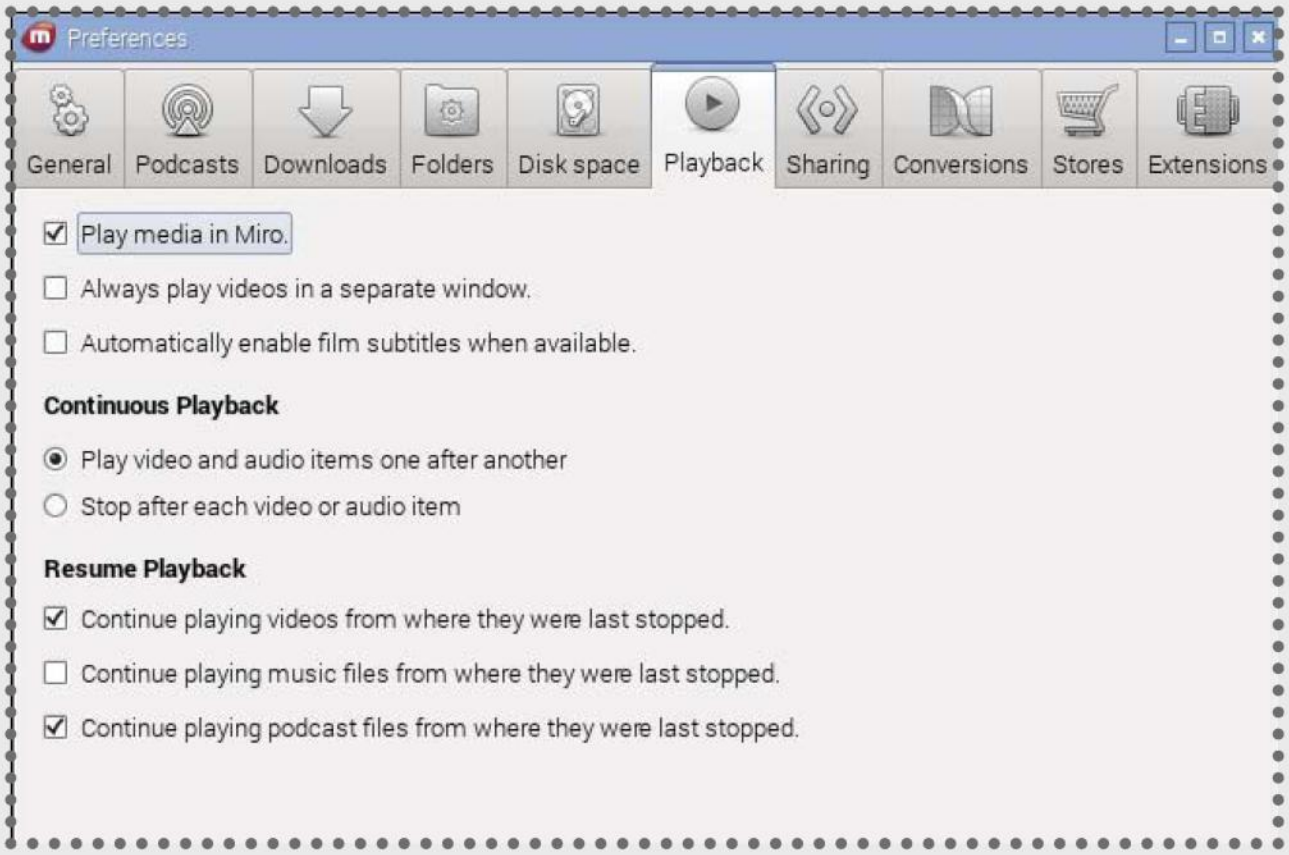
07 Add podcast feeds

In Miro, open up File>Add Podcast and then paste the podcast feed URL into the dialog that appears, clicking Create Podcast when you're done.

The more links you add, the more regularly updated content will be downloaded to your Pi-powered Miro media manager, ready to watch on demand.

Checking for new content

It is tempting to set a regular frequency for your content checking in File> Preferences> Podcasts, but note that checking too regularly is going to result in resources being hogged temporarily, which may result in an interruption if you happen to be actually watching something when Miro checks for new content. Limit polling to hourly or daily checks.



08 First time use

Remember earlier when we instructed Miro to behave a particular way upon launch? It's time to set that behaviour now, by opening the Videos view in the left-hand pane and playing the first video. Each time you boot your system, Miro will jump to this view and begin playing.

09 Avoid YouTube

As good a solution as Miro is to building a video podcast streaming center, displaying material that you're interested in on demand, it's sadly just no good for videos on YouTube. This doesn't really restrict you too much as there are plenty of other media outlets to cover, but it's worth mentioning if you're a frequent YouTube watcher. This is a shame, but shouldn't impact the way you use it – your Raspberry Pi now downloads focused content on demand!

Capture photos at night with the Pi NoIR camera

Set up and deploy a mobile solution to take night-time photos of your garden's hidden wildlife

You are probably already familiar with the Raspberry Pi camera module. However, it is also available in the NoIR edition – 'NoIR' in this instance being shorthand for 'no infrared'. Using this module, you can use the camera to take photos and video footage in the dark, similar to a night vision camera. Many of us are fortunate enough to have a wide variety of animals that visits our gardens and homes, and while we obviously see them more during the day, there are also a large number of nocturnal visitors to your garden. This tutorial shows you how to set up your camera, then add and combine an infrared light source and PIR motion sensor to trigger the camera and photograph night-time wildlife. Each photo that is captured is saved to the Pi and a 'time stamp' is also added to monitor and track the time that the animal visited your garden.

THE PROJECT ESSENTIALS

NoIR camera module
LISIPAROI LED light ring
http://bit.ly/1meQGaR
PIR Sensor
Portable battery

01 Attach the NoIR camera module

To get started, add your camera to the Raspberry Pi. It is vital that you ensure that you remove all static electric charge you may have built up by touching, say, a radiator first, as the camera can be damaged or even destroyed by static. The blue-coloured label points away from the HDMI port. Once in place, start up the Raspberry Pi as normal. Access the configuration settings using sudo raspi-config and enable the camera as shown below, then reboot.

It is always advisable to update the software, so type this into the command line:

```
sudo apt-get update
sudo apt-get upgrade
```

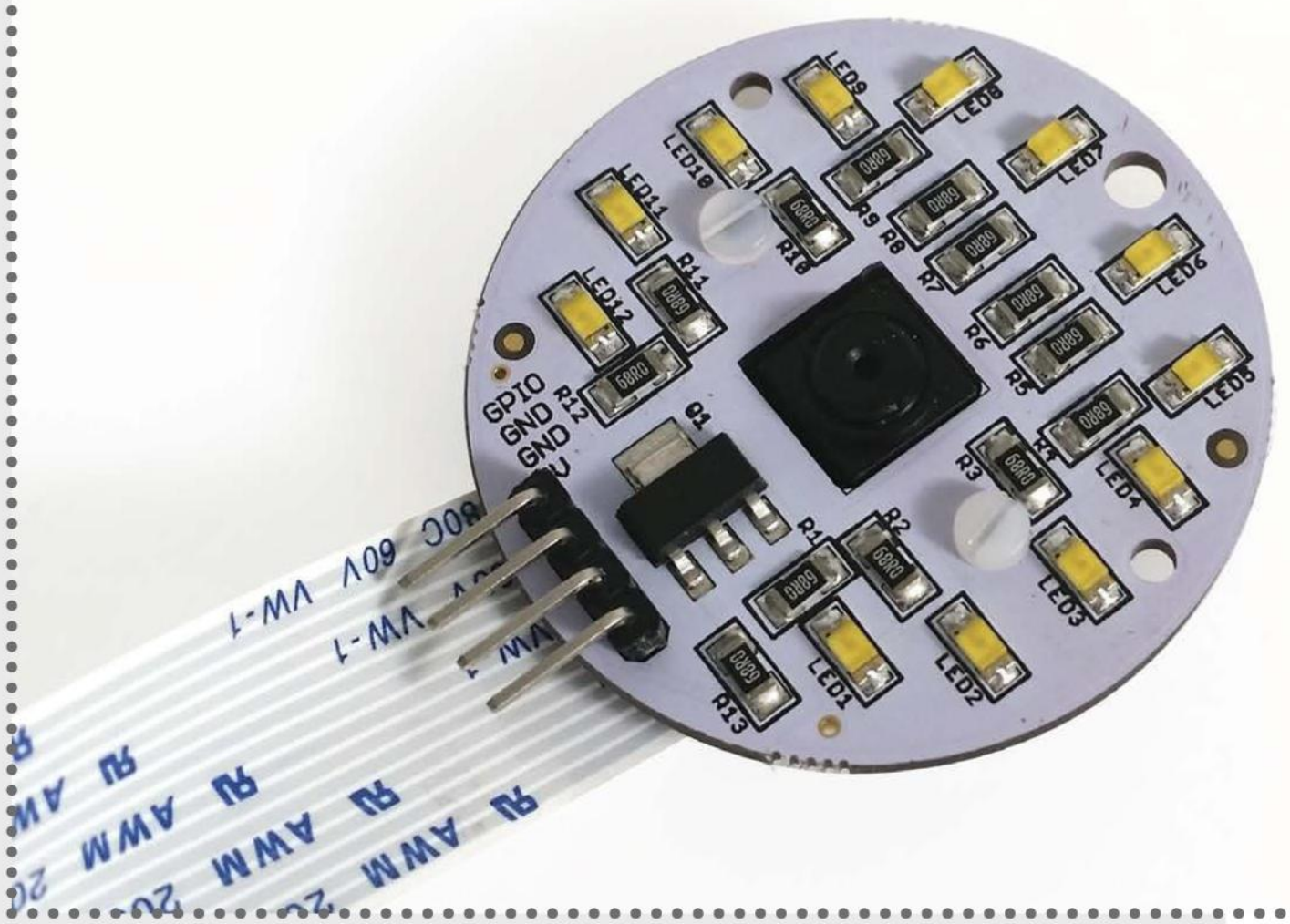
Above The NoIR camera module picks up just enough light to enable you to take some fascinating outdoor photography at night

02 Take a picture

Once the Pi is updated, test that the camera is working correctly. In the LXTerminal, enter the following: raspistill -v -o test.jpeg. When run, you will see a brief preview on the screen and a picture will be taken and saved with the file name 'test'. This file is saved in the /home/pi folder. The parameter –o is for output and this is the name of the file you save the image as. For example, raspistill –o keyboard.jpeg saves the image as a file called keyboard.

03 Using Python to take pictures

In this project, you will use Python to control the NoIR module and capture the pictures. Taking a picture with Python can be easily achieved with just a few lines of code. Open your Python editor and create the function below, then save and run. The image can be previewed using the code camera.start_preview() (line 6). To take a

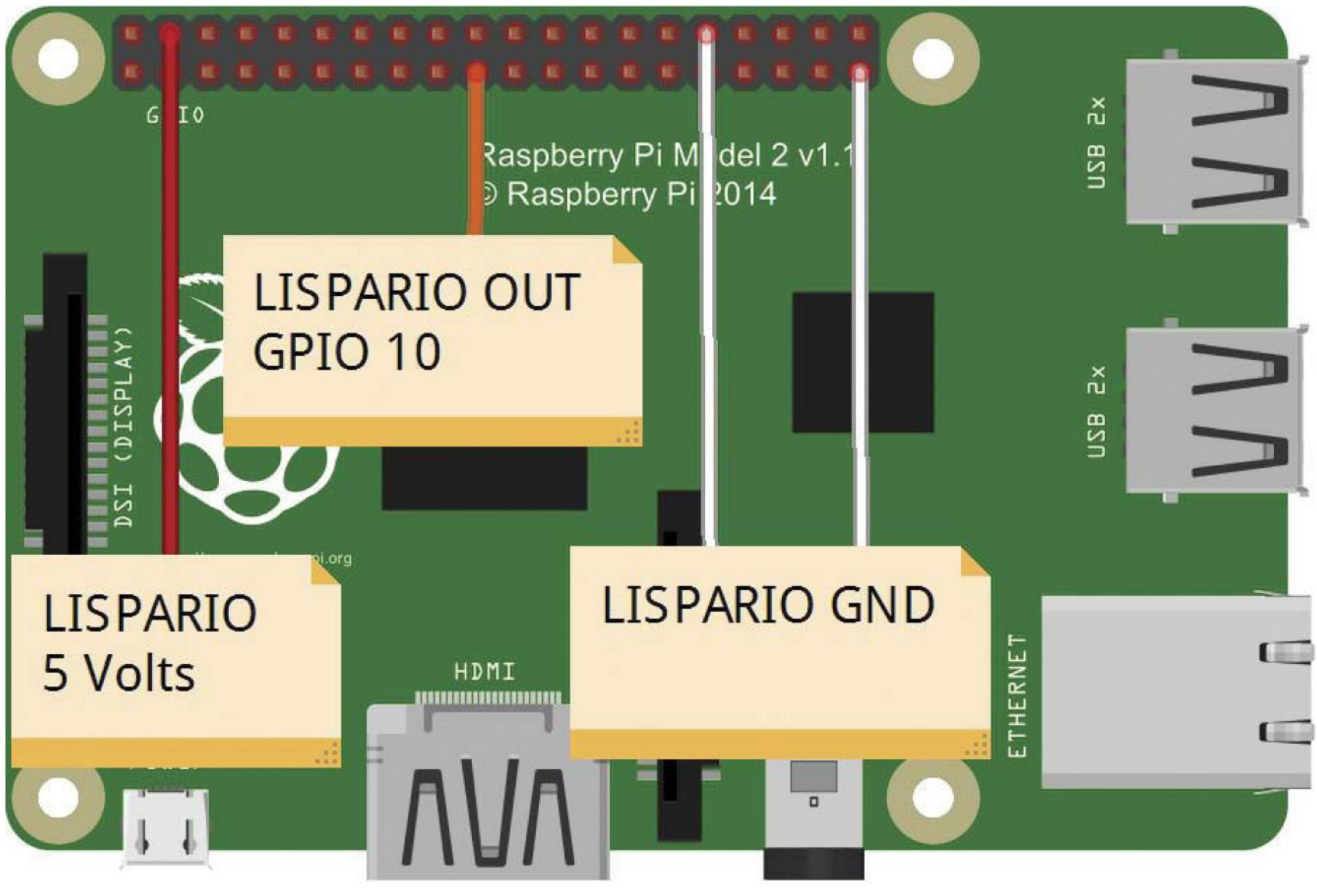


picture, use camera.capture('nature.jpg') (line 8). Replace 'nature' with the name that you wish to call the image file. This will test that the camera is working correctly with Python.

```
import time
import picamera

def Nature_selfie()
 with picamera.PiCamera() as camera:
    camera.start_preview()
    time.sleep(2)
    camera.capture(‘nature.jpg’)

Nature_selfie()
```



04 But does it work?

When you take your first photo in the dark, it may appear that the NoIR module does not work. The picture is not a night vision spy-style photo; in fact, it is probably completely black. What you need to change this is an infrared light source. The LISIPAROI is an add-on for the camera module which is designed to provide additional illumination when taking pictures or recording video in the dark. It features extra mounting points, which are perfect for using custom mounts or a gooseneck holder. There are two versions: a standard for the original camera and the NoIR version. The infrared LISIPAROI

(noted with clear LEDs) has 12 infrared LEDs arranged around the camera module to offer a wide spread of light when used in low/zero light conditions, making it perfect for capturing night wildlife activity.

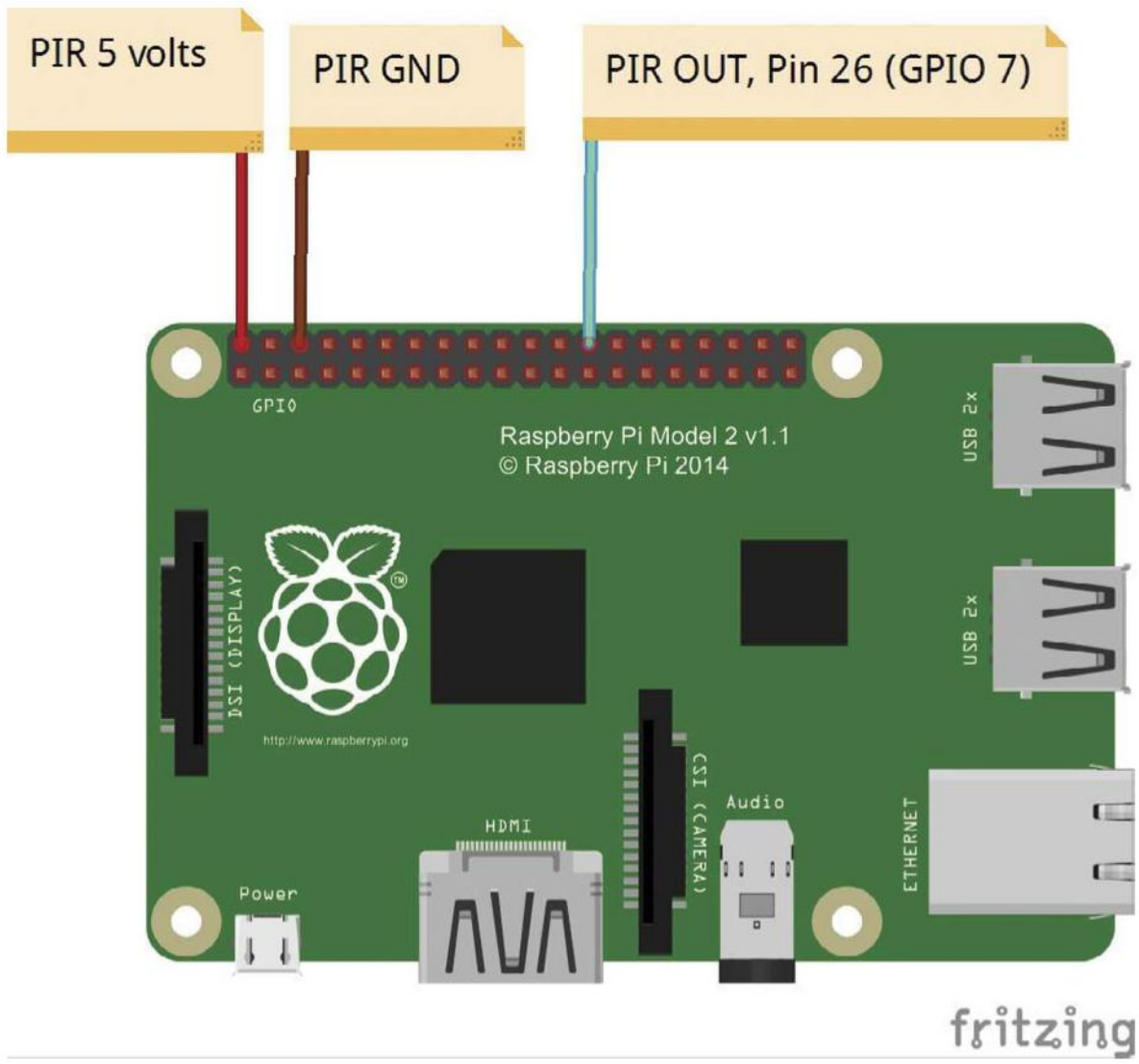
05 Connecting up the LISPARIO

Connecting the LISIPAROI is easy: simply take four female-to-female leads and attach them to the pins on the LISIPAROI. Connecting to the Raspberry Pi is easy, too: the pins required are 5V, GND, GND and the GPIO 10 pin. In this project, the code uses the BCM pin numbering system – the physical pin number is provided here for ease. The 5V is attached to physical pin 4, the ground wires go to pins 32 and 39, although there are several GND pins you can use. The final wire is the GPIO 10 pin, which is physical pin number 19 on the board. Now you have your LISIPAROI connected and you are ready to take a picture.

06 Another test

Now it's time to adapt your previous camera code to turn on the LISPARIO and capture a picture in the dark. First, import the GPIO module (line 1, below): import RPi.GPIO as GPIO. Set the GPIO numbering system to BCM with the code GPIO.setmode(GPIO.BCM) on line 4. The LISAPRIO runs on GPIO pin 10, so set this on line 5: GPIO.setup(10, GPIO.OUT). Before the picture is taken, set the output to HIGH on line 6 to turn the LEDs on: GPIO.setup(10, GPIO.HIGH). The picture is then taken and saved. On the last line, the LEDs are turned off. Check your image file – you should now have a night vision-style photo.

"Taking a picture with Python can be easily achieved with just a few lines of code"




```
import RPi.GPIO as GPIO
import time
import picamera
GPIO.setmode(GPIO.BCM)
GPIO.setup(10, GPIO.OUT)
GPIO.output(10, GPIO.HIGH)

with picamera.PiCamera() as camera:
 camera.start_preview()
  time.sleep(2)
  camera.capture('nature.jpg')
 camera.stop_preview()

GPIO.output(10, GPIO.LOW)
```

07 PIR sensor

A PIR sensor, 'passive infrared sensor', picks up the heat energy that is given off by objects. This radiation is invisible to the human eye because it radiates at infrared wavelengths. However, it can be detected by electronic devices such as the PIR. The sensor can be used to detect when a change in heat has occurred. The PIR has two dials that can be changed to adjust the settings of the PIR. The first is to adjust the 'heat' sensitivity, which will make the PIR trigger with more or less of a heat change. The second dial adjusts the rest time between the sensor stopping and restarting. This is originally set to a delay of around a few seconds.

08 Connecting the PIR sensor

Time to connect the PIR. To check this is working correctly, first remove the LISIPAROI wires. The PIR has three wires: the 5V, a ground and the out wire. Remember you are using the BCM pin numbering system, so the number stated in the code will be the GPIO pin number on the Raspberry Pi. The +5V wire connects to physical pin 2, the out connects to pin 26(GPIO 7) and the Ground connects to physical pin 6.

09 Testing the PIR

Now that the PIR sensor is connected, test that it is working correctly and adjust the settings to ensure that it triggers correctly. Open your Python editor, open a new file and enter the test code below. The important line of code is :

```
GPIO.add_event_detect(PIR, GPIO.RISING,
```

Cron

Cron, which many say stands for Commands Run Over Night, is used as a time-based job scheduler which permits you to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance programs used for administration or disk-related tasks.

```
callback=Motion_Sensing)
```

It is set to detect the GPIO rising as the heat from, say, a badger is detected by the PIR and it triggers the GPIO pin 7. The Raspberry Pi identifies that the voltage is rising and executes the callback. In this example, the callback is a function called Motion_Sensing, which when run will display the phrase "We see you" in the Python console window. When the badger moves, there is another change in heat and the PIR senses this.

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

PIR = 7
GPIO.setup(PIR, GPIO.IN)

print "Ready to find you"
time.sleep(2)

def Motion_Sensing(PIR):
 print "We see you"

try:
 GPIO.add_event_detect(PIR, GPIO.RISING,
callback=Motion_Sensing)
 while 1:
  time.sleep(100)
except KeyboardInterrupt:
 print "Quit"
 GPIO.cleanup()
```

10 Saving as a new picture

Currently, when the Pi takes a picture using the code in Step 3 it overwrites the existing file because both files have the same filename. This is not suitable for taking multiple pictures over a period of time.

To ensure the files are not over-written, create a variable called File_Number. Each time the camera is triggered this variable is incremented by a value of one. So in the example below, the first file is called Nature1.jpg, then the next files are Nature2.jpg, Nature3.jpg and so on. This saves each new photo with a different file name and won't overwrite existing images.

```
camera.capture("Nature" + str(File_Number) +
".jpg")
        File_Number = File_Number + 1
```

11 Add the time the picture was taken

You may be keen to know the time that the camera was triggered and the pictures taken, especially if the setup has been running over night. To do this, create a new variable called time_of_photo, at the start of the program, to store the current time. To retrieve the time that the photo was taken use time.asctime(time.localtime(time.time())). This will check the local time of your Raspberry Pi and save it to the variable time_of_ photo. Ensure that your Pi's clock is set correctly before you start your program.

```
time_of_photo = time.asctime(
time.localtime(time.time()) )
```

BCM number

GPIO pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off. You can also program your Raspberry Pi to turn them on or off. The GPIO.BCM option means that you are referring to the pins by the "Broadcom SOC channel" number. If you start counting the pins, this is the physical pin number. The GPIO.BOARD option specifies that you are referring to the pins by the plug number, ie the numbers printed on the board.

12 Add the time to the picture

Recording the time that the picture is taken is only relevant if you are there watching and waiting for wildlife to trigger the camera, and you can see the time value. A better solution is to use the time to add a 'time stamp' to the picture. This means that when you view each image you can see the time that the pictures were taken at the top. The line of code required is: camera.annotate_text = "time_of_photo".

13 Recap

Before you finalise the project, a quick recap on the features and setup: the NoIR camera module is attached to the Pi to capture photos in the dark. This requires an infrared light source, which is provided by the LISIPAROI. A PIR is attached and used to sense changes in heat, which then turns the LEDs on and triggers the camera to capture a picture. The current time is added to the picture for future reference. The diagram on the facing page shows the wiring solution for the LISIPAROI and PIR.

14 Taking a video

You may decide that you would prefer to take a video when the PIR is triggered; you could adapt the program to trigger and record videos of wildlife that may visit your garden or the location where your camera is. Again, Python makes it simple to record video using the code camera.start_recording('/home/pi/Desktop/evidence.jpg') to start the recording, which you can save as a file called evidence. If you want to video for, say, 20 seconds then use time.sleep(20) before stopping the recording with camera.stop_recording('/home/pi/Desktop/evidence.jpg').

```
camera.start_preview()
      camera.start_recording('/home/pi/Desktop/evidence.h264')
  time.sleep(20)
  camera.stop_recording( )
```

15 Automatically start the program

If you deploy the project outside then you will probably not have a monitor or screen attached, meaning that you cannot see what the program is doing. To sort this, set the program to start automatically when the external power supply is plugged in. First, write down where you have saved the Night Box program – for example, in the /home/pi folder, called Night_Box.py, in which case you'd use /home/pi/Night_Box.py. Double-check you've got the correct path by opening the LXTerminal and typing:

```
sudo cat /home/pi/name_of_your_script.py
```

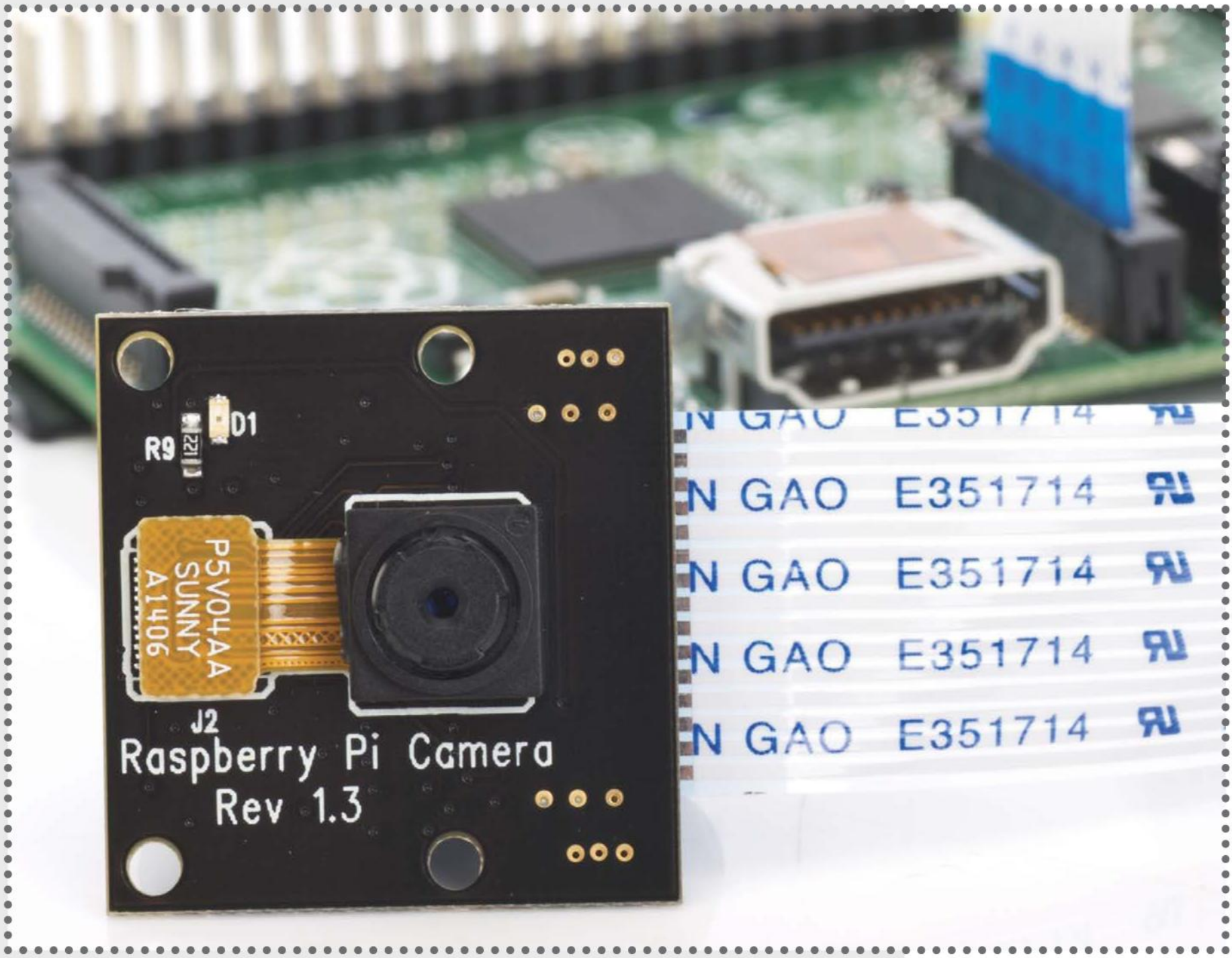
If correct, this will display the contents of your Python code. The next step is to create a Cron job by modifying the 'crontab'. To edit it, type sudo crontab –e (this will run the Cron task for all users) Scroll to the bottom of the window and then add the following line to the file:

```
@reboot python /home/pi/name_of_your_program.py &
```

The "&" at the end will run the code in the background and ensure your Raspberry Pi will boot up as normal. Save the file by hitting Ctrl+X followed by Y, and then reboot.

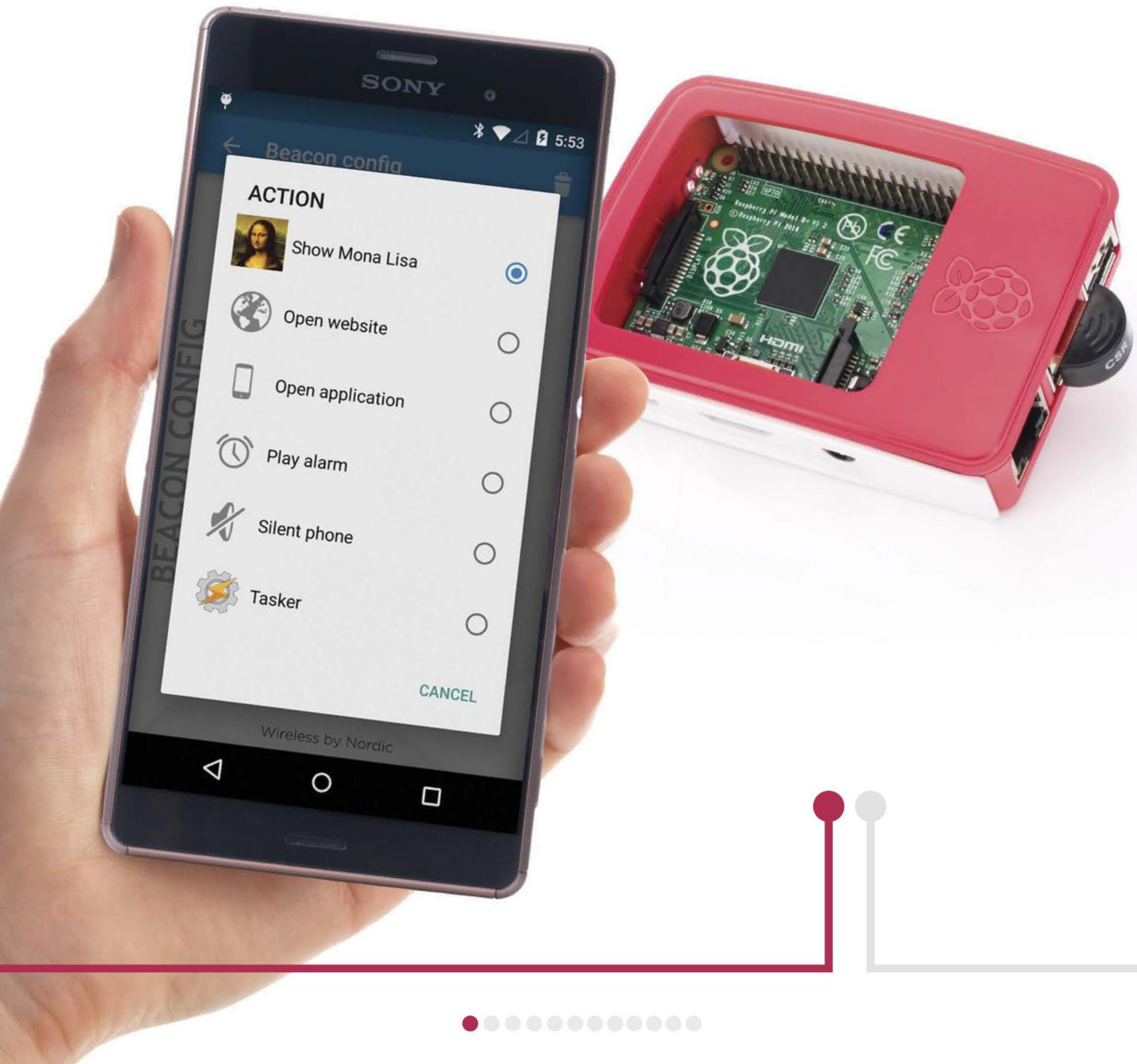
16 Deploy the Night Box

To deploy the setup effectively you will want to create a box or holder for the PIR, LISIPAROI, etc. This will protect them from the weather, especially if it rains or snows. You may want to consider a 3D-printed solution or even an old match box to hold the components. In the previous step you scheduled a Cron task to start the program automatically when the Raspberry Pi is first booted up. Find a suitable location and set up your box. Plug in your portable power supply and the Raspberry Pi will boot up and start the program. Check back the next day to see who visited your garden the previous night!



Build an iBeacon with a Bluetooth dongle

Get to grips with Apple's iBeacon technology using your Raspberry Pi as a DIY PiBeacon



The 2002 movie 'Minority Report', based on the Philip K Dick novel, is set in the archetypal dystopian future, where police react to crimes before they happen and adverts are targeted at individuals. Such targeted advertising displayed ads and messages based on your identity, interests and habits. It was seen as one of many futuristic elements to the film, but fast-forward to now, and that targeted advertising is here.

While it doesn't rely on retinal identification (yet), iBeacon from Apple uses Bluetooth and an iPhone 4S or later in close proximity to trigger an app or message. Google, meanwhile, has its own version, known as Eddystone. Raspberry Pi owners can investigate this technology in more detail by setting their devices up as 'PiBeacons' with the addition of a low-cost Bluetooth Low-Energy (BLE) USB module, available from ModMyPi at **http://bit.ly/1MtDbJC**.

THE PROJECT ESSENTIALS

USB Bluetooth Low-Energy device
http://bit.ly/1MtDbJC

Android/iOS device

Android/iOS beacon app

BlueZ

01 Prepare your Pi

To set your Raspberry Pi up as a PiBeacon, begin with a fresh, updated version of Raspbian, perhaps installed using NOOBs. From here, run the update and upgrade commands to ensure that your Raspbian version is fully up to date. When this is done, you can proceed to install the libraries required for your PiBeacon project.

Again, these will take a little bit of time to download and install. Once the libraries have installed, however, you'll be able to proceed and set up Bluetooth.

02 Set up your Pi with Bluetooth

You've probably got a Bluetooth dongle already, but before connecting, ensure that it is the correct type (see Bluetooth devices boxout on the right). Using a standard
USB Bluetooth dongle for this project will result in a lot of wasted time and frustration.

Once connected, open a terminal and enter lsusb to confirm the device is connected and detected. A suitable Bluetooth Low-Energy device will be listed as a Bluetooth 4.0 device.

Once you're happy the device is detected, create a new directory for BlueZ (the Linux Bluetooth stack), switch to the new directory and download BlueZ:

```
sudo mkdir bluez
cd bluez
sudo wget www.kernel.org/pub/linux/
bluetooth/bluez-5.11.tar.xz
```

03 Build BlueZ

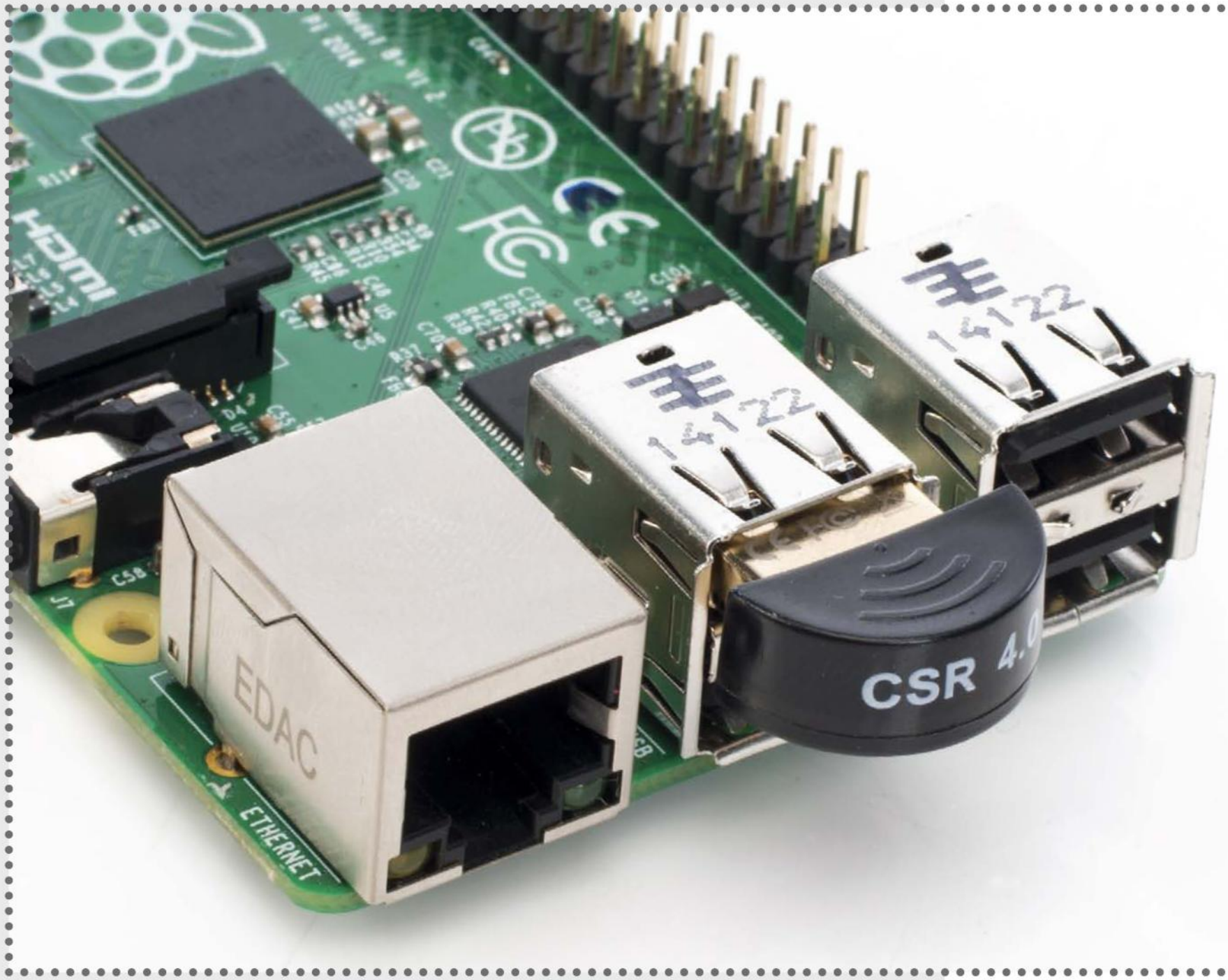
Next, remove the BLE device and build BlueZ, extracting the files and configuring the system. Note that the sudo make command will take some time to complete, so don't worry if the Raspberry Pi seems unresponsive. Just wait, and when done, input the final sudo make install, and wait again.

Once this is done, you'll need to switch off your Raspberry Pi with the usual sudo shutdown –h now command. Wait for the power light to go out, and then reconnect the Bluetooth Low-Energy USB dongle.

Follow this by reconnecting the power supply and then rebooting your Raspberry Pi:

Bluetooth devices

A Bluetooth Low-Energy (BLE) device is the key to success with this project, but if you opt for any old USB Bluetooth dongle, the chances are you won't have the success you're expecting. For the best results, order online from ModMyPi (the version 4.0: bit.ly/22npQOL). If shopping elsewhere, check the description and reviews to ensure the device is BLE (alternatively listed as "Bluetooth LE" and "Bluetooth Smart"), and Raspberry Pi compatible. Avoid the cheaply made, overpriced Bluetooth dongles sold in supermarkets!

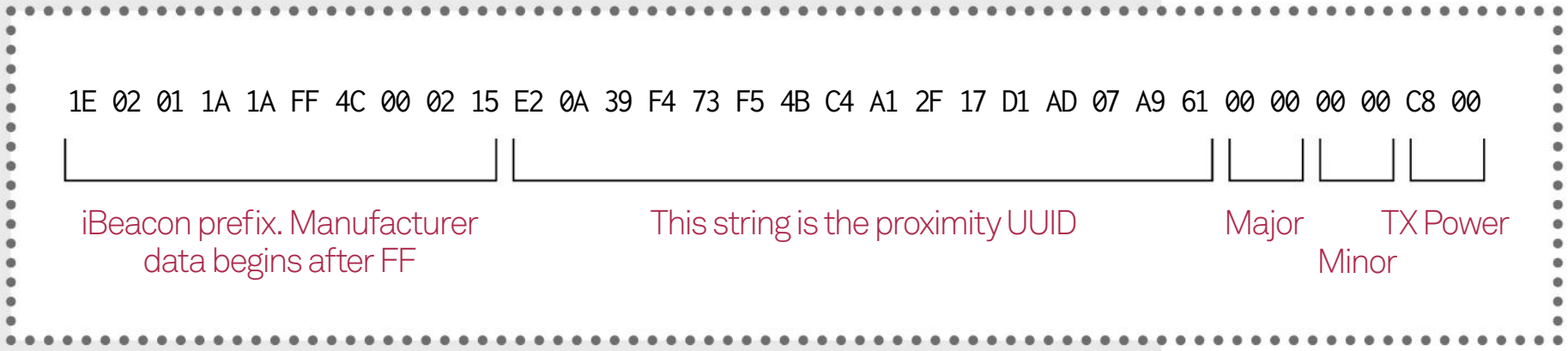



```
sudo unxz bluez-5.11.tar.xz
sudo tar xvf bluez-5.11.tar
cd bluez-5.11
sudo ./configure --disable-systemd
sudo make
sudo make install
```

04 Configure for Broadcast

With the BLE connected and the Pi restarted, enter hciconfig in the command line to configure the Bluetooth module. It will usually be listed as being "DOWN". To resolve this, input the following code. As ever, stay aware of the commands you're inputting.

Above Our PiBeacon relies on Bluetooth technology, so you'll need a dongle



Above Here's how a universally unique identifier breaks down into more meaningful chunks

In this case, the final line ends with noscan, which disables Bluetooth scanning. This is done to avoid issues with the module attempting to scan while simultaneously broadcasting. With these commands entered, enter hciconfig once more. This time, the device status will be "UP RUNNING".

```
sudo hciconfig hci0 up
sudo hciconfig hci0 leadv 3
sudo hciconfig hci0 noscan
```

05 Input a UUID

The PiBeacon is now ready. All you need to do is instruct it to advertise to other (correctly configured) devices. This simply means using the hcitool command to attract attention to the PiBeacon. Note how the universally unique identifier (UUID) below is broken into pairs. Like any such identifier, certain portions of the string (everything after "cmd") are intended to communicate particular information. For instance, after FF, manufacturer-specific data is displayed, in this case 4C 00, which is Apple's ID. The example UUID shown here can be used on your Pi, but note that it is intended for a particular app – specifically, the 99p Beacon Toolkit app in the App Store.


```
sudo hcitool -i hci0 cmd 0x08 0x0008 1E 02
01 1A 1A FF 4C 00 02 15 E2 0A 39 F4 73 F5
4B C4 A1 2F 17 D1 AD 07 A9 61 00 00 00 00
C8 00
```

06 Resolve errors

From time to time you may encounter errors with your PiBeacon, specifically in persuading the status to switch from "DOWN" to "UP RUNNING".

Restarting the Pi is no good here. Instead, you need to reset the hci device. First, enter hciconfig to check the hci device name. Usually, this is hci0.

Next, enter sudo hciconfig hci0 reset. This essentially clears everything out and lets you start again with hci. To check, input hciconfig again, where you should see that the status now reads "UP RUNNING".

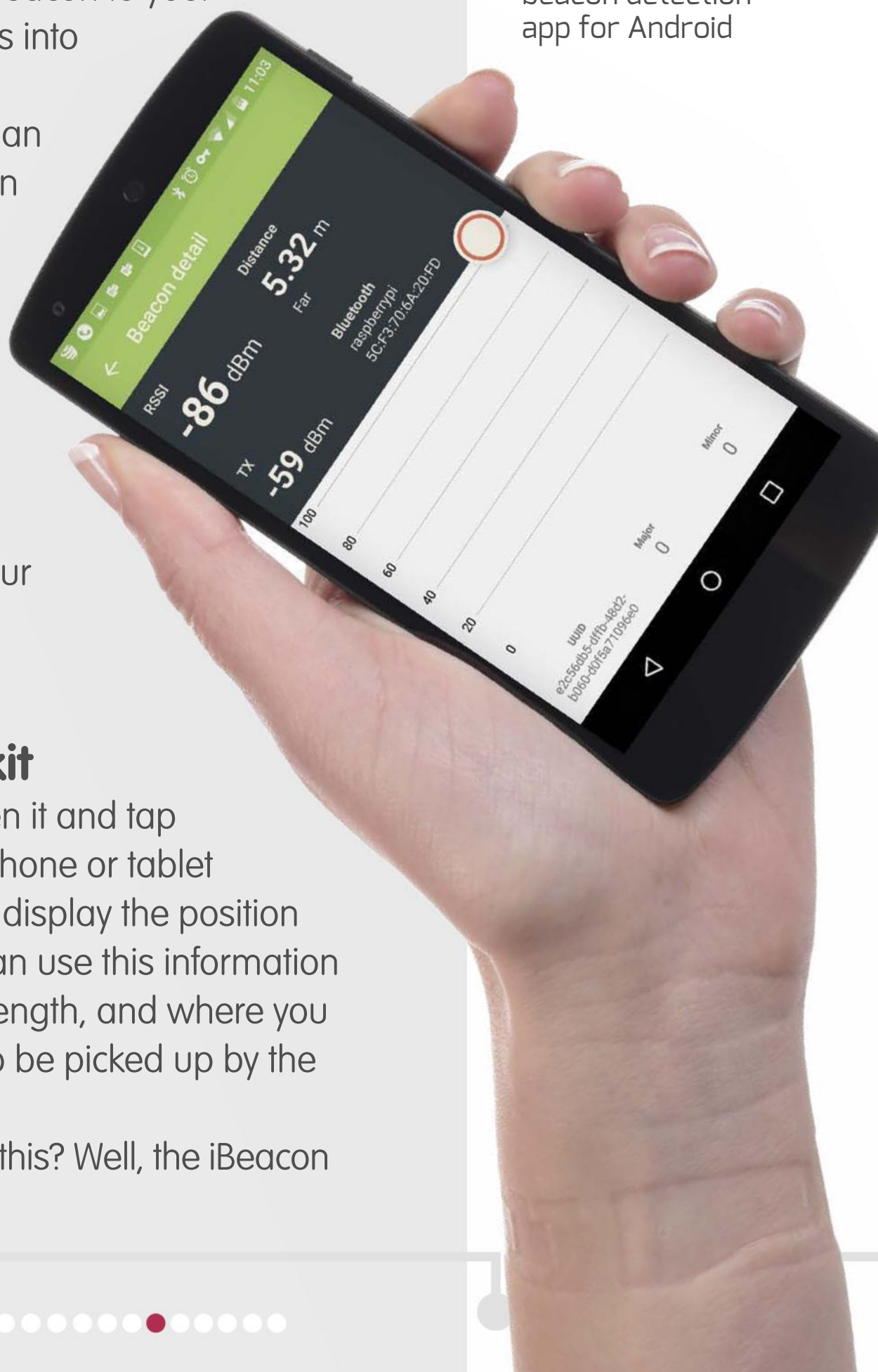
Above You'll need an app such as iBeacon Detector when you're first setting up your PiBeacon

```
sudo hciconfig hci0 reset
```

07 Test with an iOS device

Before proceeding to use your PiBeacon for a particular project, you'll need to make sure that it is broadcasting correctly. To do this, you'll need an iPad or iPhone (or even an iPod Touch) in order to detect the Scan Response Data – information (including the UUID) sent from the PiBeacon to your smartphone when it comes into proximity with it.

In addition, you'll need an app. Something like Beacon Toolkit, mentioned above, should be sufficient. We're really just looking at this stage to test the strength and range of your PiBeacon. Both of these qualities rely on the BLE device connected to your Raspberry Pi's USB port, of course.



Below iBeacon & Eddystone Scanner is another useful beacon detection app for Android

08 Using Beacon Toolkit

With the app installed, open it and tap Listen. As you move your phone or tablet around, the iOS device will display the position of the PiBeacon and you can use this information to get an idea of signal strength, and where you can expect the PiBeacon to be picked up by the intended app or service.

But how might you use this? Well, the iBeacon

technology is really like a sort of NFC, but without the close proximity of contact. This naturally has some security concerns, so be careful how you use it…

09 iBeacon scanning on Android

Various apps are available for Android users planning to communicate with their Raspberry Pi iBeacons. These include Locate Beacon, iBeacon Detector and Beacon Scanner. As well as being compatible with the iBeacon, they also provide support for Eddystone, the Android answer to iBeacon, and AltBeacon, an open specification iBeacon alternative. Pictured at the lower-left of the opposite page is iBeacon & Eddystone Scanner (aka Beacon Scanner), which will scan for a nearby beacon following a simple button press. Once the iBeacon is found, the app displays live info, and when this is tapped, a more detailed set of data.

10 PiBeacon projects to consider

Various projects for Raspberry Pi-powered iBeacons can be developed, including systems that can assist with any smart home builds you have planned. Among these include things like garage door automation, switching on a device such as lamp, or simply tracking the whereabouts of your Raspberry Pi.

Note that the vast majority of projects will work with whatever mobile device you are using, so overlook the platform-specific details of detection whenever you're researching such projects. To get you started, we'll use the remainder of this tutorial to take a look at a couple of PiBeacon projects you can build today.

"If you have an old Model A or Model A+ lying around at home, this would be ideal"

11 Making your PiBeacon smart

You can take what you've done so far further with some smart interaction between your smartphone and Raspberry Pi, but where should you start? Rather than going headfirst into a DIY task, it pays to take your time finding the right app for your smartphone, one that will enable you to launch apps based on the presence of a PiBeacon.

The tools we've looked at above are simply for detection and measurement, and the data they reveal is useful for developing trigger apps. But how do you launch an app when your phone is near a PiBeacon?

12 PiBeacon Trigger Apps

Whether you're using iOS or Android, you'll find a trigger app on the corresponding app store that can be set up to launch an app when your smartphone comes into proximity with your PiBeacon. Android users should take a look at nRF Beacon, available free from Google Play and intended for devices equipped with Bluetooth 4.0 hardware and running Android 4.3 or later.

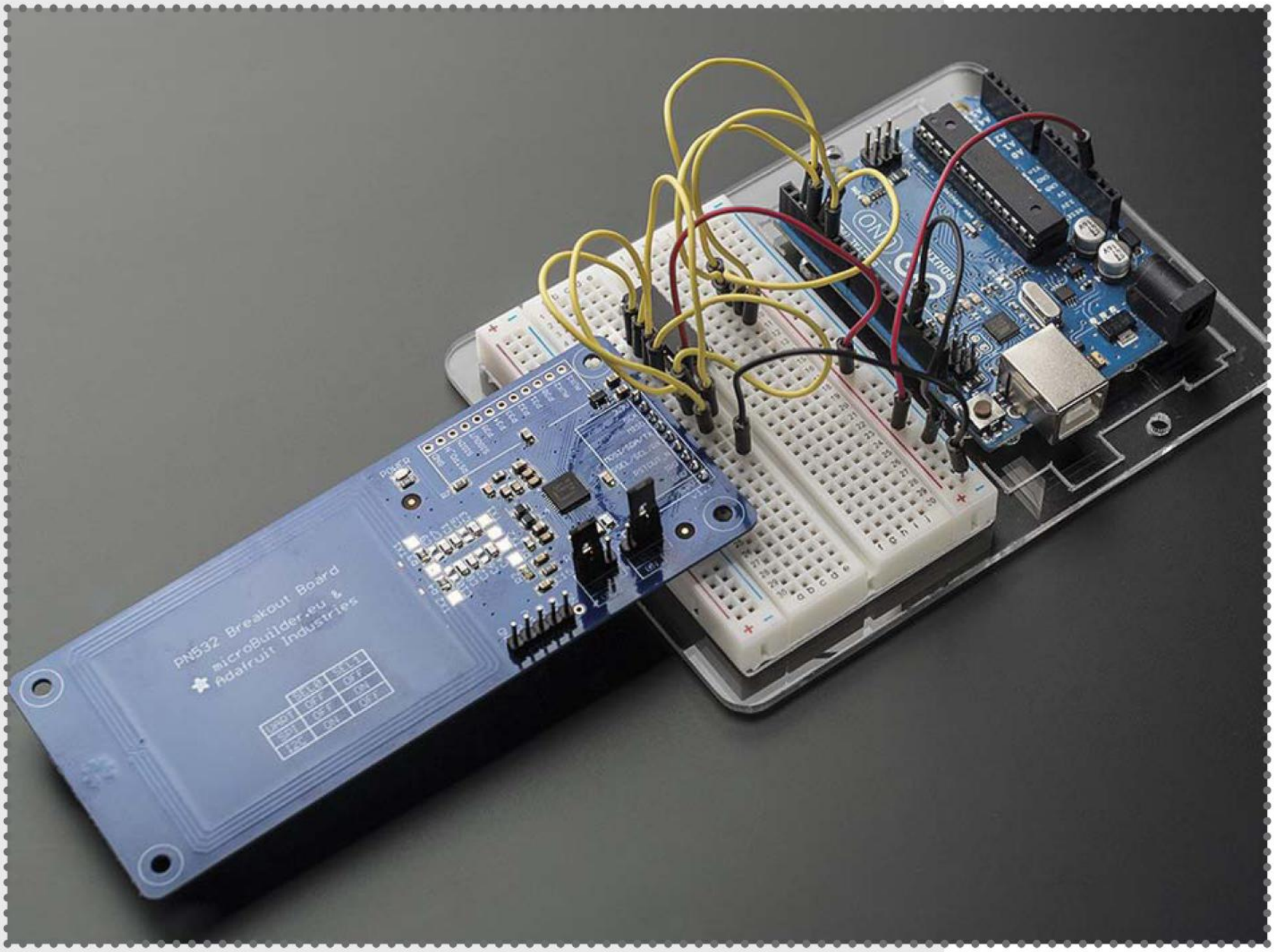
Meanwhile, iOS users should take a look at Beecon, available for £1.59 in the App Store. This app has support for IFTTT, Philips Hue, HomeKit and Lux lights integration.

13 What a PiBeacon can't do

As things stand, for better or worse, you're not in a Philip K Dick novel. This means that for the time being (at least), iBeacon technology cannot launch a web page on your phone without a third-party tool (such as those mentioned above).

Signal strength

Whether you plan on developing your own apps or projects using PiBeacon, keep in mind this one unavoidable truth: smartphones and tablets all have different antennae, something that can result in vastly different results for signal strength and proximity. For this reason, you should set triggers to occur at low proximity (that is, close to the PiBeacon); alternatively, you might employ multiple PiBeacons, or simply target specific smartphone models only.



It is not beyond the realm of possibility that such a feature might be integrated into iOS or Android in the foreseeable future, a situation that might see an advert displayed on your phone as you pass an iBeacon, perhaps including a discount for shopping at the advertising store.

14 Choosing PiBeacon vs NFC

As you may have gathered by now, there are some similarities in use for an iBeacon and NFC, especially when it comes to smart home applications. So which one should you use for your project?

Adafruit produces an NFC breakout board to add NFC functionality to the Raspberry Pi, which can

prove very useful for any project that requires physical contact between an NFC tag and a piece of hardware – we grabbed ours from ModMyPi.

And this is really where the distinction lies. If your project requires physical contact, use NFC. Otherwise, the PiBeacon solution should be adequate.

15 Preparing for smart home hardware

Although a very useful addition to smart homes, the PiBeacon isn't a miracle fix. While you might want to use it to open your garage door – perhaps with an IFTTT command send once your smartphone comes into proximity – without the right mechanical hardware connected to your Raspberry Pi, this won't be possible.

Similarly, you'll need to employ suitable electronic hardware to your PiBeacon to aid in the switching on and off of connected devices, such as a lamp. The PowerSwitch Tail II is such a device, and ships with jumper connectors for hooking it up to your Raspberry Pi GPIO.

16 Weather-proof your PiBeacon

Smart home projects using iBeacon technology may well need weather proofing. This might mean secreting your Raspberry Pi inside a suitable all-weather case, for example, or placing it somewhere sheltered from the elements.

Additionally, if no standard power supply is available, you'll need to add a battery pack. You may consider a standard portable USB power supply, or prefer to use a DIY solution, such as that

shown in issue 154 of our sister magazine Linux User & Developer. This explained how to power your Raspberry Pi with AA batteries, and covered the inclusion of a UBEC, a voltage regulator that protects the Raspberry Pi from being damaged by the batteries.

17 PiBeacon, low profile

Thanks to the fact that a PiBeacon project really only needs a power supply and USB Bluetooth BLE dongle, you can rely on the less-featured/popular Raspberry Pi devices for any PiBeacon projects.

If you have an old Model A or Model A+ laying around at home, this would be ideal. And if you're fortunate enough to have bagged a Raspberry Pi Zero for just £4, this too would be suitable. All you would need in addition is a USB to micro-B USB adaptor, which would then be connected to the Pi Zero's micro-B USB for data port.

Code a Tempest clone in FUZE BASIC Part 3

Add the finishing touches to your FUZE BASIC arcade game

Welcome to the final episode of our FUZE BASIC tutorial. Over the last two issues we've built up a retro tunnel shooter game inspired by the classic Tempest. This month, we're going to add in the last few features and tighten up our code to make this into a fully functional game.

At almost 900 lines, 73MP357 is now a fairly significant program. You will need to download the full program as well as a few sound effect and music files, all of which are available to download from this issue's FileSilo page or from **fuze.co.uk/tutorials/73MP357.zip**.

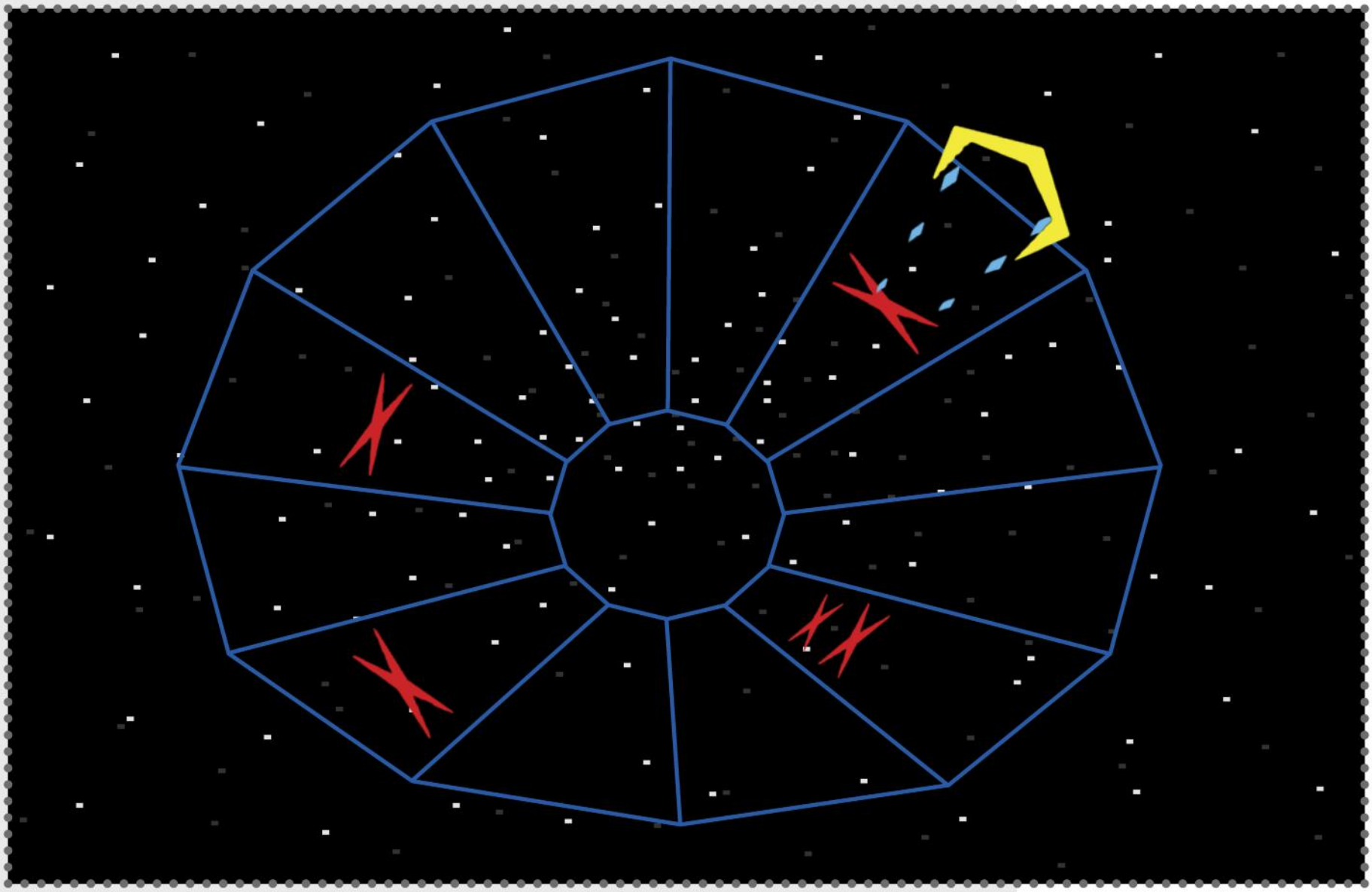
Download and decompress the files, then save the resulting folder to your FUZE BASIC folder. This final version is exactly that: the finished article. We now have levels, enemies, explosions, music and sound effects, scoring, collision and an attract screen. One more thing: FUZE BASIC has been updated somewhat since the last tutorial and as such we highly recommend you update it.

THE PROJECT ESSENTIALS

FUZE BASIC V3

fuze.co.uk/getfuzebasic

73MP357PART3.fuze



01 Main loop

Start FUZE BASIC and either load (with F8) 73MP357.fuze or copy and paste the code straight into the editor (F2 switches between the editor and immediate mode).

The first thing you might notice is that we have tidied things up significantly compared to the previous versions. As much as possible, we have compartmentalised each section so it is easier to follow and debug. It is also far easier to add new things. The main program now looks like this:

```
// 73MP357 by Luke Mulcahy - FUZE Technologies Ltd
PROC initialise
PROC attract
```

```
CYCLE // start of the main loop
 frameStart = TIME // makes a note of
the time so we can see how long a single
frame takes
 CLS2
 IF speedCount > speedCounterMax THEN //
only do anything once a certain time has
passed
   speedCount = 0
   PROC keys
   PROC updatePowerUp
   PROC warp
   PROC lasers
   PROC enemies
   PROC updateStars
   PROC updatePlayer
   PROC drawPlayingfield
   PROC updatePerspective
   IF warp = FALSE THEN // we don't update
some things when the warp effect is
active
    PROC drawPowerUp
    PROC drawlasers
    PROC drawEnemies
   ENDIF
   IF destroyed = FALSE THEN PROC
drawPlayer
   PROC checkProgress
   PROC updateDebris
   PROC updateGameInfo
   UPDATE // update and display the screen
   PROC speedLimit
 ENDIF
```

```
 speedCount = speedCount + 1
REPEAT // the end of the main loop
END
```

02 Procedural languages

Rather than go through the entire program line by line, we'll take on each individual procedure individually and explain what is going on.

Firstly, though, FUZE BASIC is a procedural language, not an object-orientated one. This means we include everything we require in the one program rather than referencing 'classes' or libraries of external functions.

While most modern languages are based around object programming techniques, the original procedural-based style is very easy to follow and understand, but on the downside it means we generally have to write everything in the one program which can turn it into quite a beast.

Procedures therefore allow us to break the program up into sections similar in some ways to objects but, most importantly, into smaller and far neater segments of code that are much easier to debug.

So, to the first PROCedure then!

03 Initialise game variables

The DEF PROC initialise section initialises the game environment variables and loads the logo sprite, sound effects and music files. You will see quite a few DIM statements initialising variable arrays; DIM stands for Dimension. We declare an array variable and specify its dimensions. Then rather than using

simple cellOne=1, cellTwo=5, cellThree=9, etc, we can declare DIM cell(5), then cell(0)=1, cell(1)=5, cell(3)=9 and so on. These can easily be populated with a simple LOOP. The difference is we now only need one variable to store any number of values. You can even create multidimensional arrays – chessBoard(8, 8), for example, could be used to hold the positions of the pieces: chessboard(0, 0)=Castle, chessboard(0, 1)=Knight and so on. Arrays are widely used across programming languages and are fundamental; it is worth getting to know them.

The reason we're using so many arrays is that we need a lot of variables to store all the information about the stars, the lasers and even the debris from explosions. Every single star, bullet and piece of debris has a set of coordinates, an angle and speed of travel, etc. Sound effects and the main soundtrack are loaded and given identifiers, and we end by loading a high score file so the main score can be recorded.

04 Make an attract screen

A feature of retro arcade machines, the attract screen is a simple graphic and/or animation used to entice passers-by into popping a pound into the slot to play. The DEF PROC attract section of code displays the initial title logo along with a starfield effect for good measure. The starfield is updating by calling the updateStars process that we defined previously.

05 Control the game's speed

Let's take a look at the speed limiter function as it is used for both the main loop and the attract routine. Just before the end of the main loop, PROC speedLimit

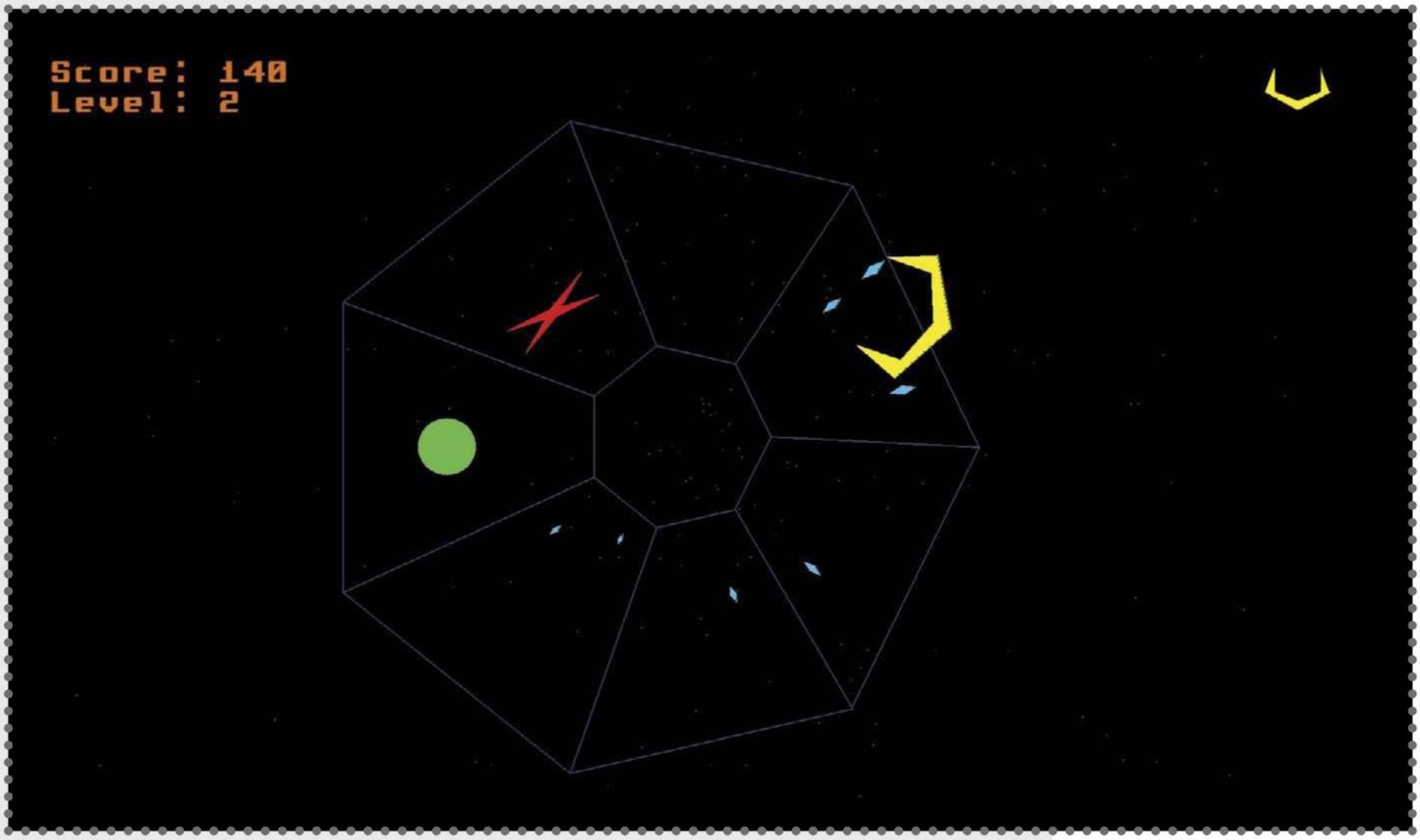
is called to check to see if we should run the main loop again or not. If our speedCount has not reached the speedCounterMax limit then it ignores the main loop and goes round again until it does. Only then is the main loop executed. This allows us to control the entire game speed with a single speedLimit variable (set at the very end of the program).

```
frameStart = TIME
CLS2
IF speedCount > speedCounterMax THEN
... (the main code)
PROC speedLimit
ENDIF
speedCount = speedCount + 1
REPEAT

DEF PROC speedLimit
frameEnd = TIME
timeTaken = frameEnd - frameStart
speedCounterMax = gameSpeed - timeTaken
IF speedCounterMax < 0
THENspeedCounterMax = 0
ENDPROC
```

06 Check for keyboard input

The DEF PROC keys block is straightforward. The routine checks to see if Z (or A), X (or D) or the Spacebar (or Enter) have been pressed. If the rotation keys (Z and X) are pressed, the player's angle variables are adjusted accordingly, whereas the Spacebar initiates a laser to be fired. Before firing it makes sure there aren't too many and which type,



standard or particle, to fire.

Left Once everything is correctly set up, you'll be greeted with this intro page

07 Update the power-up

The updatePowerUp procedure checks to see if we already have a power-up on-screen, and if not there's a 1-in-200 chance that one will be created. The first power-up always gives you particle lasers, then after this either a score boost or an extra life is awarded. To see if we have collected the power-up, we just check if it has reached the outer edge and is in the same segment as the player: IF powerupAngle > playerAngle - (gap / 2)' and 'IF powerupAngle < playerAngle + (gap / 2).

08 Activate the warp drive

The warp procedure checks to see if a warp has been activated and, if so, modifies the inner and outer radii so they zoom off the screen. The inner radius is

increased by a smaller amount than the outer so you get a warp drive effect.

09 Check the lasers

One of the shorter sections of our final code listing, DEF PROC lasers is just a quick check (a simple IF statement) to see if we are running particle lasers or not. The fire speed is adjusted accordingly, using hard-coded values rather than more variables.

10 Stagger the spawns

Next up is DEF PROC enemies. Here we use a counter variable, enemyDelay, to determine how long to wait between each new enemy. Early in the game the delay is quite long so you won't likely see more than one at a time, but as you progress then it is possible to see ten or more at once.

11 Update the starfield

To create the illusion of flying through space, we have a playing field that displays objects like enemies and the background stars moving outwards from the centre. The stars are handled with DEF PROC updateStars: the FOR CYCLE loop counts from zero to maxStars. A distance is added at the current angle using COS and SIN. If the star goes off the screen then it is repositioned in the centre of the screen with a small random offset, so they don't all start in the dead centre.

12 Update the player

Next, the player is drawn (in yellow) using a simple polyPlot function inside the updatePlayer procedure.

Then there are a few steps to determine the movement, so when you go from one segment to another it glides over and doesn't just jump from segment to segment. The FN lerp function is used to determine the movement step

13 Calculate the lerp

The lerp function (DEF FN lerp) takes three numbers, of which the first two are the start and end point of something. It could be the distance between two points, a range of colours or the score, both before and after. The third number is used to work out an equal step between the first two. The function is used here to ensure that we get equal and therefore smooth movement steps.

```
DEF FN lerp(a, b, c)
result = a + c * (b - a)
= result
```

14 Playing field and perspective

Next up, we draw the playing field. This is handled with the drawPlayingfield procedure. Each segment is drawn using three lines – top, side and bottom – and repeats around until completed. The Gap variable determines the size of each segment. Moving straight on to the next procedure, since it's related, updatePerspective is a nice little routine. This simply checks to see which corner of the screen the player is in and then shifts the centre origin a bit so that everything is then drawn from that centre point. It is a very effective way to create a pseudo 3D effect.

15 Draw the power-up

The drawPowerUp procedure checks to see if the power-up is within the outer radius and, if so, modifies its brightness depending on how close to the edge it is. Lerp is used again to figure out an even step from one colour to the next and smoothen the transition from the centre outwards.

16 Draw the lasers

DEF PROC drawlasers is huge, but actually there's a lot of repetition so it's not as tough as it looks. The first section works out, depending on whether we're using a standard or particle laser, the position of each individual laser. We then perform pretty much the same calculations on each instance. Notice that, when working on the particle version, there are two (left and right) to deal with.

Following this we check to see if any of the enemies have come into contact with a laser. This is done by comparing the laser distance and segment to that of each enemy – if they match, then BOOM! The explosion takes place and debris goes everywhere! The enemies are reduced by one, a score is added and sound effects are played depending on laser type.

17 Draw the enemies

Now for the enemies: DEF PROC drawEnemies. As you will now be getting used to, the FN lerp is used to maintain smooth movement steps. It really is a very cool little function, that one. The position is calculated and then the enemy is plotted. We then check to see if it comes into contact with the player and again, if so,

then debris flies about, the player's lives are reduced by one and we check to see if it is game over. If we're not out of lives then we warp the level off the screen and start afresh.

18 Draw the player

Firstly, the c and s variables are used to store the current COS and SIN results. This is so we don't have to use tmpPlayerAngle + 90 + playerTurn against every draw calculation. The u variable is the distance from the centre of the screen. The player is then drawn using a polyPlot function.

```
 DEF PROC drawPlayer
 COLOUR = Yellow
 c = COS (tmpPlayerAngle + 90 +
playerTurn)
 s = SIN (tmpPlayerAngle + 90 +
playerTurn)
 u = gHeight / 80
 polyStart
 polyPlot ((u * 5 * c) - (u * 5 * s) +
playerX, (u * 5 * s) + (u * 5 * c) +
playerY)
 polyPlot ((u * 5 * c) - 0 + playerX, (u *
5 * s) + playerY)
 polyPlot (0 - (u * -2 * s) + playerX, (u
* -2 * c) + playerY)
 polyPlot ((u * -5 * c) - 0 + playerX, (u
* -5 * s) + playerY)
 polyPlot ((u * -5 * c) - (u * 5 * s) +
playerX, (u * -5 * s) + (u * 5 * c) +
playerY)
```

```
 polyPlot ((u * -7 * c) - 0 + playerX, (u
* -7 * s) + playerY)
 polyPlot (0 - (u * -4 * s) + playerX, (u
* -4 * c) + playerY)
 polyPlot ((u * 7 * c) - 0 + playerX, (u *
7 * s) + playerY)
 polyEnd
 ENDPROC
```

19 Display remaining lives

Our next procedure, DEF PROC plotPlayerLives, helps us to display the number of lives that the player has left at the top-right of the screen. This routine positions the location just below the top with GHEIGHT and then draws the lives from left to right using polyPlot commands.

20 Check progress

Now we check to see how far we've got with the checkProgress block. Every time we kill an enemy, the progress variable is increased, and if we kill enough then we surpass the advance variable, which warps us off to the next level. A score bonus is also awarded.

```
 DEF PROC checkProgress
 IF progress = advance THEN
  playSample (sfx(7), 4, 0)
  oldScore = score
  score = score + ((10 * scoreBoost) *
level)
  scoreTime = 1
  level = level + 1
```

```
    warp = TRUE
    advance = advance + 1
    enemyDelay = enemyDelay + 240 /
targetfps
  ENDIF
  ENDPROC
```

21 Update the debris

Starfields need starjunk, so next we have a very simple updateDebris procedure. Each piece of debris is updated with its new position and then plotted in a random colour.

22 Update the scores

DEF PROC updateGameInfo is responsible for displaying the current score and level. Would you believe it, yet another use for FN lerp! This time it works out the value required to equally increment the score each time any points are awarded. This is how the score increases as if it is counting up the numbers and not just adding a complete number each time like a simple addition.

23 Start the game

The setup procedure is called every time you start a new level, either from the attract screen or after a warp is initiated. All level variables are reset – not the score or difficulty, but pretty much everything else is reset or cleared ready to start again.

Then, about half way down, the WHILE intro < radius CYCLE statement starts a loop in which the playing field is zoomed into the screen. The player is

displayed, and score and lives plotted. Lasers and enemies are reset and we're off!

24 Take it to the next level

And there you have it. There's still plenty of scope for improvement. It could include more enemies, a smart bomb, more audio tracks, a jump feature and a proper end sequence with a high score table, for example. We hope you have enjoyed playing with 73MP357 and that you do go on to make many improvements – we'd love to see them if you do, so please send them in to us here at LU&D or straight to the FUZE website.

We also hope you've been intrigued by what can be achieved with FUZE BASIC. We don't expect you to keep using BASIC forever, but if it has given you the programming bug and now you want to climb the language ladder then we have done our job. Our expert recommends starting with C++; it is still the most widely-used coding language in the world and most other languages stem from it or are at least similar. If you can code in C++ then you're pretty much set and will easily be able to adapt to just about any other language. Good luck!

To find out more about FUZE BASIC and the FUZE in general please visit **www.fuze.co.uk**

Multitasking with your Pi

Learn how to add multitasking to your own Python code – perfect for those with multiple projects on the go

The majority of programmers will learn single-threaded programming as their first computational model. The basic idea is that instructions for the computer are processed sequentially, one after the other. This works well enough in most situations, but you will reach a point where you need to start multitasking. The classical situation for writing multi-threaded applications is to have them run on a multi-processor machine of some persuasion. In these cases, you would have some heavy, compute-bound process running on each processor. Since your Raspberry Pi is not a huge 16-core desktop machine, you might be under the assumption that you can't take advantage of using multiple threads of execution. This isn't true, though.

There are lots of problems that map naturally to the multiple thread model. You may also have IO operations that take a relatively large amount of time to complete. In these cases, it is well worth your programming effort to break your problem down into a multi-threaded model. Since Python is the language of choice for the Raspberry Pi, we will look at how you can add threads to your own

Python code. For those of you who have looked into multi-threaded programming in Python, you may have run into the GIL (Global Interpreter Lock) before. This lock means that only one thread can actually be running at a time, so you don't get true parallel processing. But on the Raspberry Pi, this is okay. We just want to use a more natural programming paradigm for certain problems where it makes sense.

The first bit of code we need is to import the correct module. For this article, we will be using the threading module. Once it is imported, you have access to all of the functions and objects that you would need to write your code. The first step is to create a new thread object with the constructor:

```
t = threading.Thread(target=my_func)
```

The Thread object takes some function that you have created, my_func in the above example, as the target code that needs to be run. When the thread object has finished its initialisation, it is alive but not running. You need to explicitly call the new thread's start() method. This will begin running the code within the function handed to the thread.

You can check to verify that this thread is alive and active by calling its is_alive() method. Normally, this new thread will run until the function exits normally. The other way a thread can exit is if an unhandled exception is raised. Depending on your experience of parallel programs, you may already have some ideas on what types of code you want to write. For example, in MPI programs, you typically have the same overall code

running in multiple threads of execution. You use the thread's ID and a series of if or case statements to have each thread execute a different section of the code. To do something similar, you can use something like:

```
def my_func():
id = threading.get_ident()
if (id == 1):
  do_something()
thread1 = threading.Thread(target=my_func)
thread1.start()
```

This code works in Python 3, but the get_ident() function doesn't exist in Python 2. Threading is one of those modules that is a moving target when moving from one version of Python to another, so always check the documentation for the version of Python your are coding for.

Another common task in parallel programming is to farm out time-intensive IO into separate threads. This way, your main program can continue on with the core work and all of the computing resources are kept as busy as possible. But how do you figure out if the child thread is done yet or not? You can use the is_alive() function mentioned above, but what if you can't continue without the results from the child thread? In these cases, you can use the join() method of the thread object you are waiting on. This method blocks until the thread in question returns. You can include an optional parameter to have the method time-out after some number of seconds. This allows you to not get trapped into a thread that will never return due to some error or code bug.

Now that we have more than one thread of execution happening at the same time, we have a new set of issues to start worrying about. The first is accessing global data elements. What might happen if you have two different threads that want to read, or even worse write, to the same variable in global memory? You can have situations where changes to the value of variables can get out of sync with what you were expecting them to be.

These types of issues are called race conditions, because the different threads are racing with each other to see in what order their updates to variables will happen. There are two solutions to this type of problem. The first is to control access to these global variables and only allow one thread at a time to be able to work with them. The generic term describing this control is to use a mutex to control this access. A mutex is an object that a thread needs to lock before working with the associated variables. In the Python threading module, this object is called a lock. The first step is to create a new Lock object with:

```
lock = threading.Lock()
```

This new lock is created in an unlocked state, ready to be used. The thread interested in using it must call the acquire() method for the lock. If the lock is currently available then it changes state to the locked state and your thread can run the code that is meant to be protected. If the lock is currently in a locked state, then your thread will sit in a blocked state, waiting for the lock to become free. Once you are done with the protected

"It is worth the effort to break your problem down into a multi-threaded model"

code, you need to call the release() method to free the lock and make it available for the next thread. As an example, you could control a variable containing the sum of a series of results with code like:

```
lock.acquire()
sum_var += curr_val
lock.release()
```

This can lead to another common issue in parallel programs: deadlocks. These issues occur when you have multiple locks that are associated with different global variables. Say you have the variables A and B, and the associated locks lockA and lockB. If thread 1 tries to get lockA then lockB, while thread 2 tries to get lockB then lockA, you could have the situation where they each get their first requested lock, and then wait forever for the second requested lock.

The best way to avoid this type of bug is to code your program very carefully. Unfortunately, people are only human and messy code can creep in. You can try and catch this kind of bad behaviour by including the optional timeout parameter when you call the acquire() method. This tells the lock to only try and get the lock for some number of seconds. If the timeout is reached, the acquire method returns. You can tell whether or not it was successful by checking the returned value. If it was successful, acquire will return True. Otherwise, it will return False.

The second way you can deal with data access is by moving any variables that you can to within the local scope of the individual threads. The essential idea is

that each thread would have its own local version of any required variables that nobody else can see. This is done by creating a local object. You can then add attributes to this local object and use them as local variables. Within the function being run by your thread, you would have code that looks like:

```
my_local = threading.local()
my_local.x = 42
```

The last topic we will look at is synchronising your threads so that they can work together effectively. There will be times when a number of threads will need to talk to each other after working on their separate parts of a particular problem. The only way they can share their results is if they have all finished calculating their individual results. You can solve this problem by using a barrier, which each thread will stop at until all of the other threads have reached it. In Python 3, there is a barrier object that can be created for some number of threads. It will provide a point where threads will pause when they call the barrier's wait() method.

Because you need to explicitly tell the barrier object how many threads will be taking part in the barrier, this is another area where you can have a bug. If you create five threads but create a barrier for ten threads, it will never reach the point where all of the expected threads have reached the barrier. The other synchronisation tool is the timer object. A timer is a subclass of the thread class, and so takes a function to run after some amount of time has passed. As with a thread, you need to call

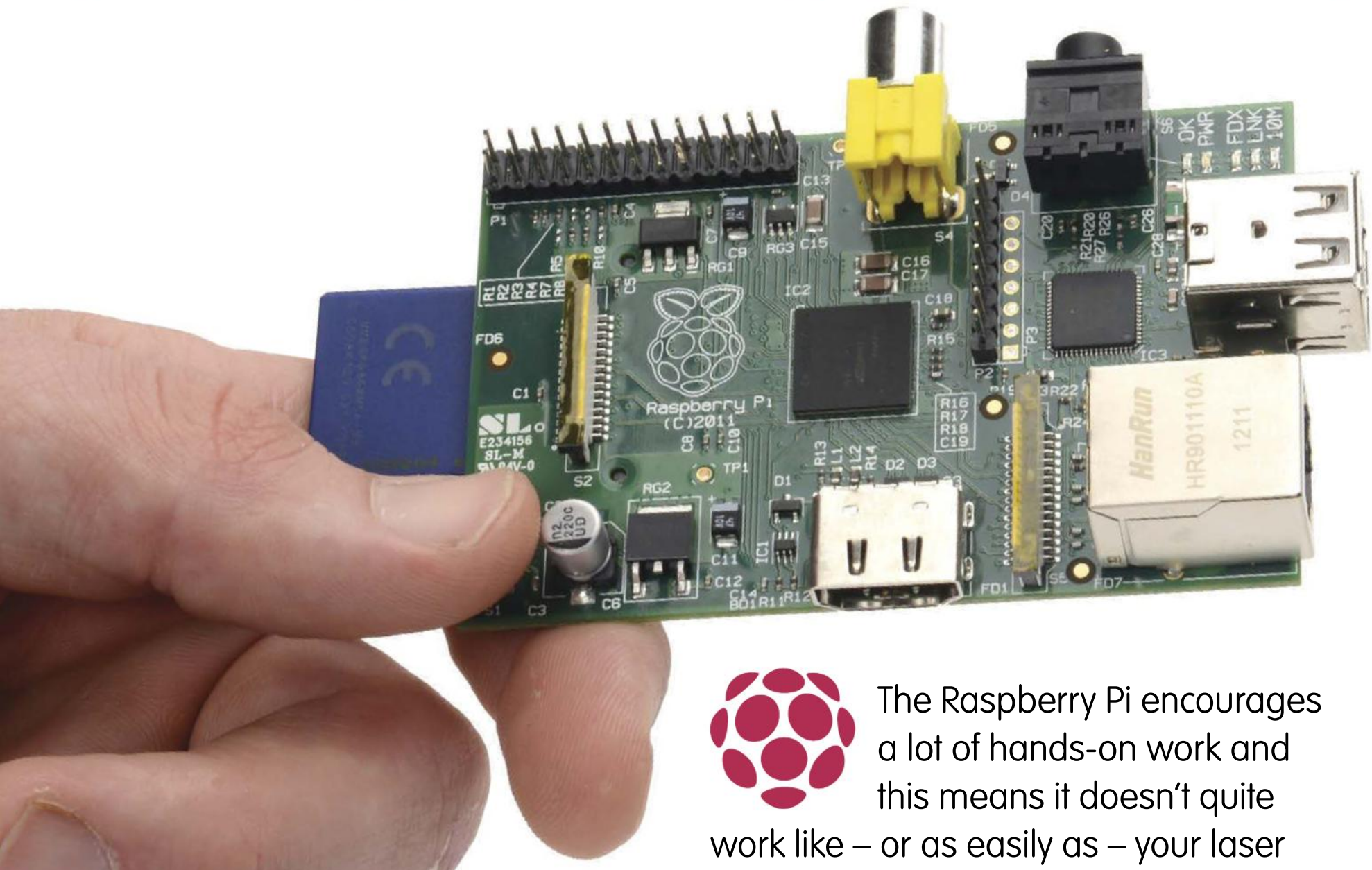
the timer's start() method in order to start the countdown to when the function gets executed. A new method, cancel(), allows you to stop the countdown of the timer if it hasn't reached zero yet.

You should now be able to have your code running even more efficiently by farming out any time intensive parts to other threads of execution. In this way, the main part of your program can remain as reactive as possible to interaction with the end user and you can keep all parts of your Raspberry Pi as busy as possible.

Talking Pi

Join the conversation at...

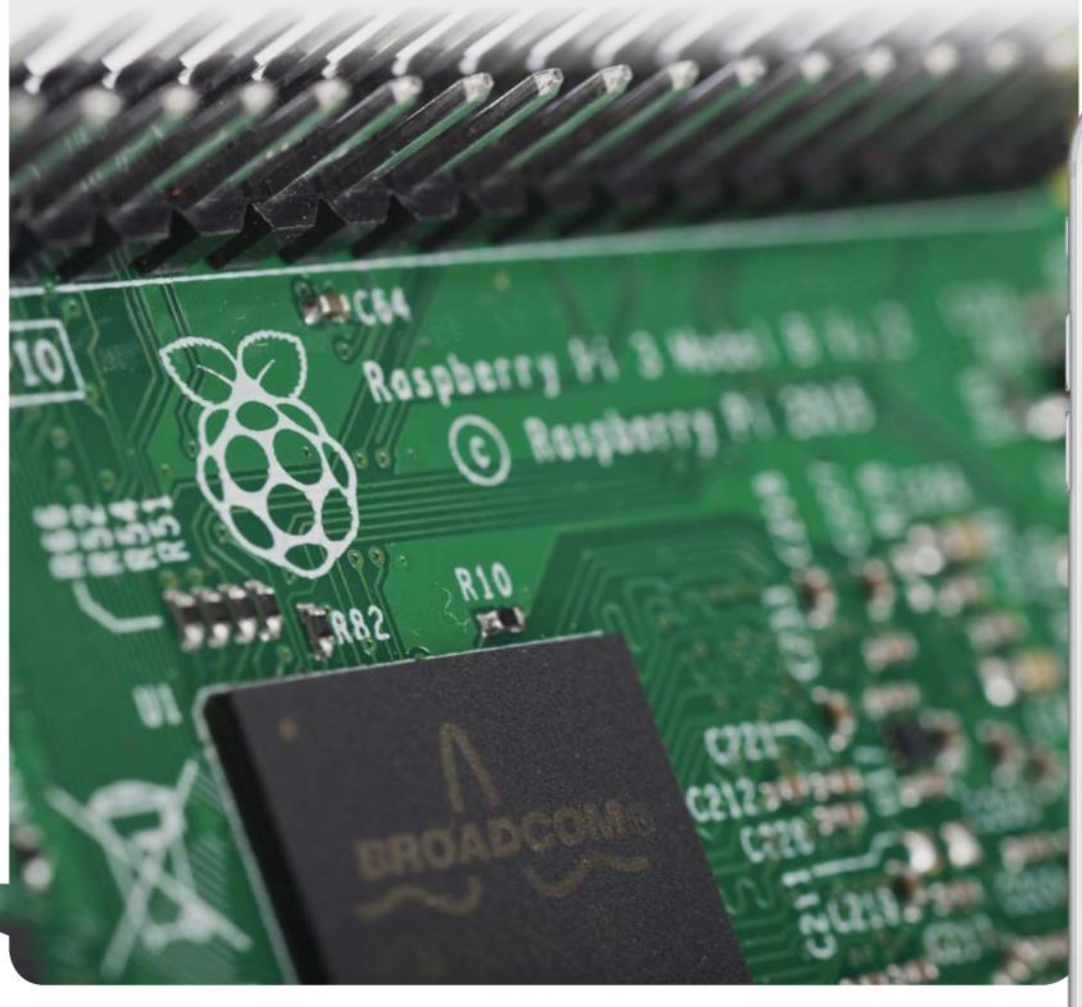
@linuxusermag | Linux User & Developer | @ RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easily as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

I've been using a breadboard to practice making circuits and connections for my Pi but I don't think it works!
Ro via email

You haven't said how your breadboard doesn't work, Ro, but we think we might have the solution for you. There are power strips that run through the middle of a breadboard. Sometimes, however, they don't run all the way across but are split in the middle. With some breadboards you can tell fairly easily, as the blue and red lines that run horizontally along the top have gaps in them in the middle of the board, to show where the power strips stop. But some breadboards, sneakily, don't have this and the lines are continuous, even though the power strips aren't! Try avoiding the holes in the very middle and see if that helps.



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

I updated my Pi ages ago and now Bluetooth doesn't work...
Jack via email

There was an update a few months ago that caused some problems with Bluetooth. Luckily it's a fairly easy fix, even if you don't want to do a fresh installation. Open a terminal window and enter **sudo apt-get install pi-bluetooth**, then reboot your Pi. If the Pi says that you already have the package installed but it's not working, try **sudo apt-get --reinstall install pi-bluetooth** and reboot. It's worth avoiding the use of a touchscreen during this process, as it can interfere.

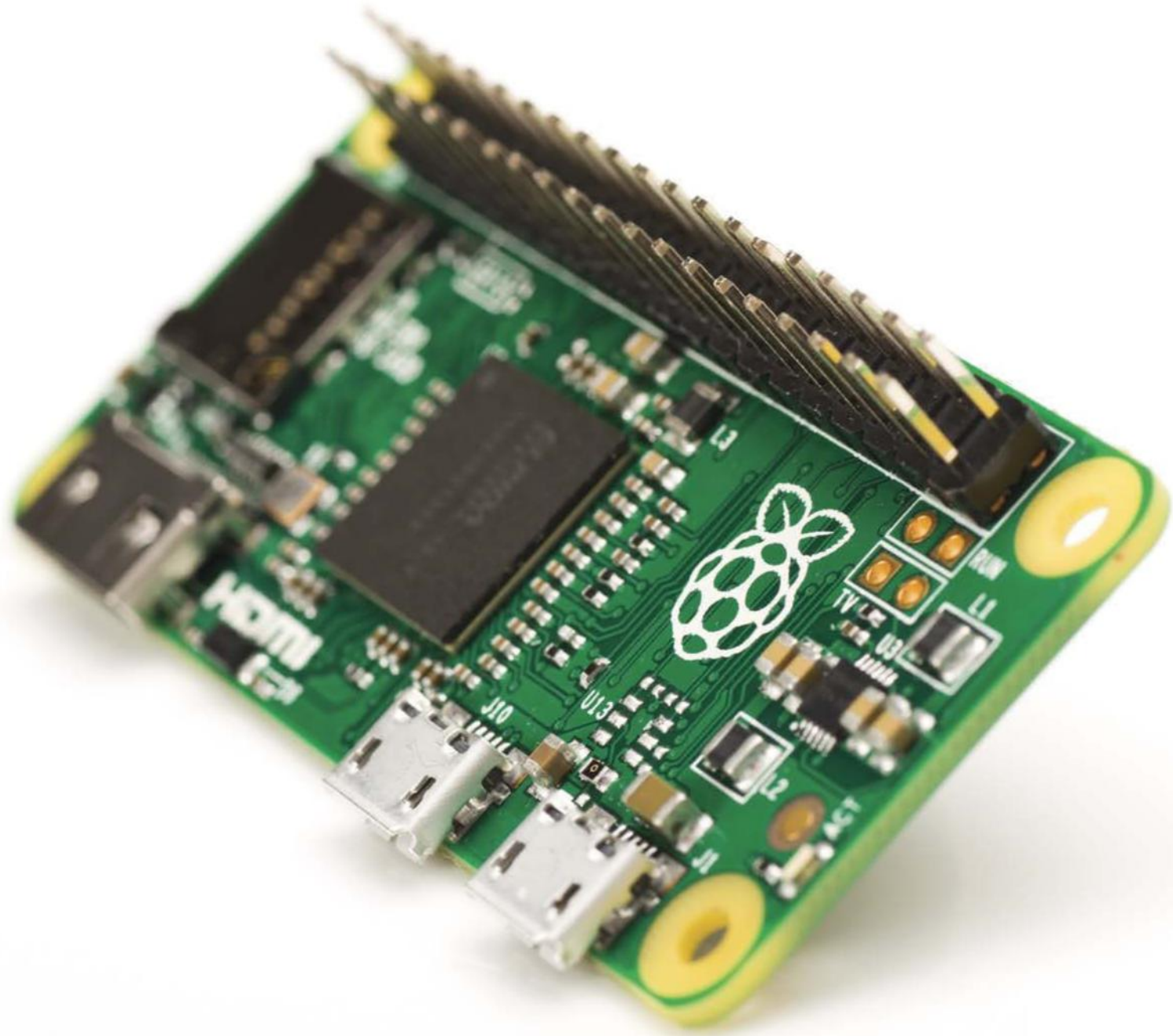
My keyboard isn't behaving like a regular US keyboard; why?
Shawn via email

That'd be because Raspbian and NOOBS default to UK English, which has a slightly different keyboard layout (Don't ask us why, it's annoyed Brits for decades). It's easy enough for y'all to fix (*Stop that immediately – Ed*) – open up a terminal in Raspbian and type **sudo raspi-config**, then choose Internationalization>Keyboard Setup. If your keyboard isn't listed just choose a generic one (101, 102 or 104), and scroll down to Other. Under the Country of Origin menu, select English (US). Complete everything else it asks you to, then reboot your Pi. You should find everything where you expect it.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Set up your Pi Zero



Get this issue's source code at:
www.linuxuser.co.uk/raspicode